TECHNIQUES FOR EFFICIENT MONTE CARLO SIMULATION. VOLUME II. RANDOM NUMBER GENERATION FOR SELECTED PROBABILITY DISTRIBUTIONS.

E. J. McGrath, et al

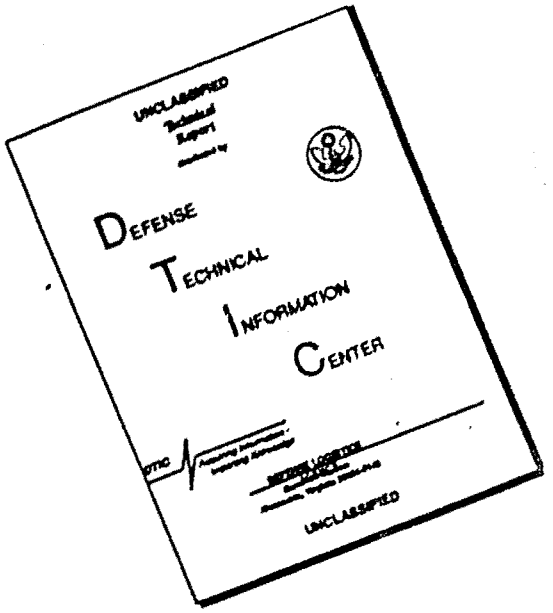
Science Applications, Incorporated

Prepared for:

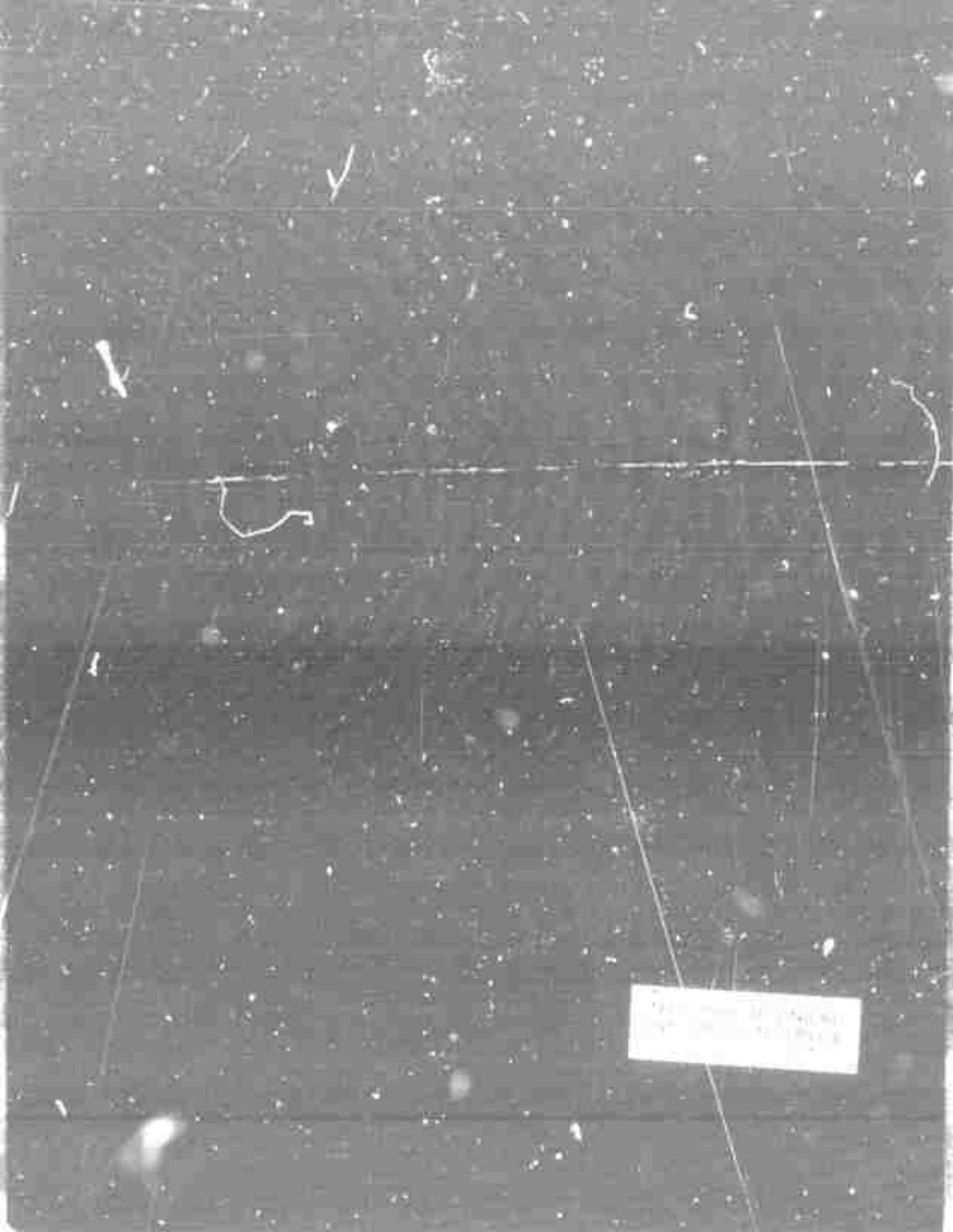
Office of Naval Research

March 1973

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.



SAI-73-909-LJ

TECHNIQUES FOR EFFICIENT MONTE CARLO SIMULATION

VOLUME II

RANDOM NUMBER GENERATION FOR SELECTED PROBABILITY DISTRIBUTIONS

Scientific Officer, Office of Naval Research (Code 462)

J. R. Simpson

Principal Investigator

E. J. McGrath

Co-Author

D. C. Irving

SCIENCE APPLICATIONS, LA JOLLA, CALIFORNIA
ALBUQUERQUE • ANN ARBOR • ARLINGTON • BOSTON • CHICAGO • HUNTSVILLE • LOS ANGELES
PALO ALTO • ROCKVILLE • SUNNYVALE • TUCSON

P.O. Box 2351, 1200 Prospect Street, La Jolla, California 92037

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Science Applications, Inc. 1250 Prospect Street La Jolla, California 92037	Unclassified
	2b. GROUP

3 REPORT TITLE

Techniques for Efficient Monte Carlo Simulation
Volume II: Random Number Generation for Selected Probability Distributions

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Final Report

5 AUTHOR(S) (First name, middle initial, last name)

Elgie J. McGrath, David C. Irving

6 REPORT DATE	7a. TOTAL NO OF PAGES	7b. NO OF REFS
March 1973	~~134~~ 120	7

8a CONTRACT OR GRANT NO	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-72-C-0293	SAI-72-590-LJ
b. PROJECT NO NR364-074/1-5-72	
c Code 462	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10 DISTRIBUTION STATEMENT

Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY
	Office of Naval Research (Code 462) Department of the Navy Arlington, Virginia 22217

13 ABSTRACT

Algorithms for efficient generation of random numbers from various probability distributions are presented, in both a flowchart form and as a sample Fortran subroutine. Twenty-two different distributions, including all commonly encountered discrete and continuous functions, the Weibull, Johnson, and Pearson families of empirical distributions, and histogram distributions, are covered. The general techniques to apply in deriving a random number selection scheme for an arbitrary distribution are discussed. A machine-independent subroutine for generating uniform random numbers is also described.

DD FORM 1 NOV 65 1473

Security Classification

ABSTRACT

Algorithms for efficient generation of random numbers from various probability distributions are presented, in both a flowchart form and as a sample Fortran subroutine. Twenty-two different distributions, including all commonly encountered discrete and continuous functions, the Weibull, Johnson, and Pearson families of empirical distributions, and histogram distributions, are covered. The general techniques to apply in deriving a random number selection scheme for an arbitrary distribution are discussed. A machine-independent subroutine for generating uniform random numbers is also described.

CONTENTS

FIGURES

EXECUTIVE SUMMARY

Monte Carlo simulation is one of the most powerful and commonly used techniques for analyzing complex physical problems. Applications can be found in many diverse areas from radiation transport to river basin modeling. Important Navy applications include analysis of antisubmarine warfare exercises and operations, prediction of aircraft or sensor performance, tactical analysis, and matrix game solutions where random processes are considered to be of particular importance. The range of applications has been broadening and the size, complexity, and computational effort required have been increasing. However, such developments are expected and desirable since increased realism is concomitant with more complex and extensive problem descriptions.

In recognition of such trends, the requirements for improved simulation techniques are becoming more pressing. Unfortunately, methods for achieving greater efficiency are frequently overlooked in developing simulations. This can generally be attributed to one or more of the following reasons:

- Analysts usually seek advanced computer systems to perform more complex simulation studies by exploiting increased speed and/or storage capabilities. This is often achieved at a considerably increased expense.

- Many efficient simulation methods have evolved for specialized applications. For example, some of the most impressive Monte Carlo techniques have been developed in radiation transport, a discipline that does not overlap into areas where even a small number of simulation analysts are working.

- Known techniques are not developed to the point where they can be easily understood or applied by even a small fraction of the analysts who are performing simulation studies or developing simulation models.

This document is the second of three volumes which present techniques and methods for developing effective Monte Carlo simulations. Each volume is essentially a self-contained discussion of useful techniques which can be applied in reducing computation effort in one of the following three major aspects of Monte Carlo simulation:

- Selecting Probability Distributions - Volume I
- Random Number Generation For Selected Probability Distributions - Volume II
- Variance Reduction - Volume III

The purpose of these volumes is to provide guidance in developing Monte Carlo simulations that accurately reflect the behavior of various characteristics of the system being simulated and are most efficient in terms of computational effort. The basic intent is to provide understanding of the concepts and methods for reducing analysis and computational effort as well as to serve as a practical guide for their application. The volumes have been prepared primarily for the systems analyst and computer programmer who has a basic background and experience in simulation and elementary statistics. Thus, the material is presented so as to not require extensive knowledge of statistical techniques or [illegible] literature search. However, it is assumed the reader is familiar with the fundamentals of Monte Carlo methods, techniques in modeling, and elementary statistics.

1. INTRODUCTION

In developing any Monte Carlo simulation, it is necessary to generate random numbers from the stochastic models used. In Volume I, the process and techniques of selecting probability models for the simulation were presented. The objective of this volume is to provide a convenient source of efficient and simple random number generators for all the probability distributions considered in Volume I. To this end flow charts and FORTRAN listings of these random number generators are provided here as well as descriptions of the techniques employed.

It is the purpose of this document to provide a convenient mechanism to select and implement these random number generators without having to resort to an understanding of the underlying concepts used in their development. Accordingly, the remainder of this report has been organized as follows:

- SECTION 2, "Efficiency Comparison of Random Number Generators," demonstrates improvements in running times expected from using the techniques developed here over those commonly used. This section has been included to provide an appreciation for the magnitude of improvements possible in using the techniques described herein.

- SECTION 3, "Generation of Random Numbers from Selected Distributions," provides algorithms defined by flow diagrams and standard Fortran subroutines that can be applied directly. This section is introduced with a convenient summary table defining where in the section a specific algorithm can be found.

- Appendix A, "Fundamental Considerations for Generation of Random Numbers," describes the fundamentals on which random number generation techniques for arbitrary distributions can be developed.

- Appendix B, "MIRAN - A Machine Independent Package For Generating Uniform Random Numbers," describes a uniform random number generator that can be used on any machine that does not have a reliable generator or on several different machines where identical random numbers are to be generated for comparison and cross checking.

Before proceeding it must be recognized that a "good" uniform random number generator is generally assumed to be available to the user. This is often not the case, although most computers today have uniform random number generators included as part of the system software. Unfortunately, many of the uniform random number generators in current use do not adequately approximate randomness to be sufficient for all Monte Carlo calculations. To alleviate this difficulty, a machine independent package for generating uniform random numbers is provided (Appendix B).

2. COMPARISON OF RANDOM NUMBER GENERATION PROCEDURES

The improvements in calculational efficiency realized by using the random number generation techniques provided here depend on the particular problem. However, by utilizing these techniques, near optimum results can be assured.

It is of interest to compare the random number generation techniques presented here with those commonly used to generate random numbers. This comparison was performed during the course of the study for several distributions, and it was found that improvements in computer time of factors varying from 2 to 5 were possible. Results for a few of the more common distributions are shown in Table 2.1 which compares the running times of the preferred techniques with those commonly used. For example, consider the normal (or Gaussian) distribution. The usual procedure is to generate 12 random numbers uniformly distributed over the interval $[0, 1]$ say $R_1, \ldots, R_{12}$, and determine

$$R_N = \sum_{i=1}^{12} R_i - 6 \quad .$$

By virtue of the central limit theorem,[6] R_N is approximately distributed according to the normal distribution. Assembly language time on a Univac 1108 was 105 microseconds per calculation using this approach. Procedures studied here were the rejection technique (see Appendix A) and a technique developed by Marsaglia.[5] The corresponding running times were respectively 74 and 30 microseconds. Not only are the running times significantly reduced, but also the more efficient ones presented here are exact (within machine roundoff errors).

Similar results were obtained with the exponential distribution where the Marsaglia technique gave a reduction in running times of a factor greater

than three (Table 2.1). The standard method used is the inverse (see Appendix A). The rejection method is discussed in Appendix A and the Marsaglia method is reported in Ref. 3.

As implied above, there are several methods that may be used to generate random numbers for a given distribution. However, where alternate approaches could be identified or developed, comparisons were made and the most efficient procedure selected. These generators are presented in the next section.

It should be noted that the more efficient techniques are slightly more complex to program; however, the slight additional effort involved generally pays off substantially in computer time.

TABLE 2.1

Running Time Comparisons Random Number Generators For The Normal and Exponential Distributions[b]

Distribution	Commonly Used Technique	Rejection[a] Technique	Marsaglia[a] Technique
Exponential	64	29	19
Normal (Gaussian)	105	74	30

[a]See Appendix A for a brief description of these techniques.

[b]All times in microseconds of UNIVAC 1108 Assembly Language time.

3. GENERATION OF RANDOM NUMBERS FROM SELECTED DISTRIBUTIONS

In this section, efficient algorithms are presented for a large number of probability distributions. These are summarized in Table 3-1 which gives the name of the distribution, the theoretical form, parameters in the distribution to be specified by the user, other random number generators used, and where the particular routines or algorithms can be found in this section of the report. Also shown under the name of the distribution is the FORTRAN subroutine name assigned to the random variable.

Once a distribution of interest has been identified, it is only necessary to define the values of the parameters indicated and to implement the algorithm from the specified pages of this section. In the subroutines, the parameters are represented by mnemonics which should be recognizable. For example, SIG is used to represent σ and SIGSQ to represent σ^2. In some places the mnemonic starts with an A to provide a floating point value such as ALAM for λ .

It will be noted that certain distributions rely on other distributions to generate random numbers. For example, generation of random numbers for the Rayleigh distribution requires random numbers from an exponential distribution. The exponential distribution in turn depends on a uniform random number generator. Based on the frequent requirement for the uniform, exponential and normal distribution, it is usually convenient to provide a basic random number generation package consisting of subroutines to generate uniform, exponential, and normal random variables as an integral part of any complex simulation program. Throughout this section these three random number generation subroutines will appear as UNFRN(R), EXPRN(R), and ANRMRN(R), respectively. where R is a dummy function

TABLE 3.1

Efficient Algorithms for a Large Number of Probability Distributions

Name of Distribution (Function Title)	Functional Form	Parameters To Be Specified	Other Random Number Generators Used	Location of Algorithm to Generate Random Numbers	
				Subsection	Page
Uniform (UNFRN)	$\frac{1}{b-a}$; $a \le x \le b$	a, b	None	3.1	10
Exponential (EXPRN)	$\frac{1}{\lambda} e^{-[(x-\epsilon)/\lambda]}$; $x \ge \epsilon$, $\lambda > 0$	λ, ϵ	Uniform	3.2	12
Normal (ANMRN)	$\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$	μ, σ	Uniform, Exponential	3.3	14
Binomial (KBINOM)	$\binom{n}{k} p^k (1-p)^{n-k}$; $k = 0, 1, \ldots, n$	n, p	Uniform, Exponential	3.4	17
Multinomial (MULNOM)	$\binom{n}{k_1 k_2 \cdots k_m} p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_m^{k_m}$; $p_1 + \ldots + p_m = 1$, $k_1 + k_2 + \ldots + k_m = n$	m, n, $p_1, \ldots, p_n$	Uniform	3.5	22
Poisson (KPOIS)	$e^{-\lambda} \frac{\lambda^k}{k!}$; $\lambda > 0$, $k = 0, 1, \ldots$	λ	Uniform	3.6	24
Hyper-geometric (KHYPRG)	$\frac{\binom{M}{k}\binom{N-M}{n-k}}{\binom{N}{n}}$; $N > M$, $k = 0, 1, \ldots, M$	M, N, n	Uniform	3.7	26
Geometric (KGEOM)	$p(1-p)^{k-1}$; $k = 1, 2, 3, \ldots$	p	Uniform, Exponential	3.8	28
Pascal (also called negative binomial) (KPASCL)	$\binom{n+k-1}{k} (1-p)^n p^k$; $k = 0, 1, \ldots, n$	n, p	Uniform, Exponential	3.9	31

TABLE 3.1 (Continued)

Name of Distribution	Functional Form	Parameters To Be Specified	Other Random Number Generators Used	Location of Algorithm to Generate Random Numbers	
				Subsection	Page
Cauchy (COCHRN)	$\frac{1}{\sigma\left[1+(x-\mu)^2\right]}$; $-\infty < x < \infty$	μ	Uniform	3.10	34
Rayleigh (RAYLRN)	$\frac{x}{\sigma^2} e^{-x^2/2\sigma^2}$ $x \geq 0$	σ	Exponential	3.11	36
Gamma (GAMRN)	$\frac{\lambda^\eta}{\Gamma(\eta)} x^{\eta-1} e^{-\lambda x}$ $x \geq 0$ $\eta, \lambda > 0$	λ, η	Uniform, Exponential	3.12	38
Beta (BETARN)	$\frac{1}{b-a} \frac{\Gamma(\gamma+\eta)}{\Gamma(\gamma)\Gamma(\eta)} \left(\frac{x-a}{b-a}\right)^{\gamma-1} \left[1-\frac{x-a}{b-a}\right]^{\eta-1}$ $a \leq x \leq b$ $\eta, \gamma > 0$	γ, η, a, b	Gamma	3.13	41
Pareto (PRTORN)	$\lambda \epsilon^\lambda x^{-\lambda-1}$; $x > \epsilon$	λ, ϵ	Uniform	3.14	43
Log-normal (ALNMRN)	$\frac{1}{\sigma(x-\epsilon)\sqrt{2\pi}} \cdot \exp\left[-\frac{1}{2\sigma^2}(\ln(x-\epsilon)-\mu)^2\right]$ $x \geq \epsilon$	ϵ, μ, σ	Normal	3.15	45
Folded Normal (FNRMRN)	$\frac{1}{\sigma\sqrt{2\pi}} \left[e^{-(x-\mu)^2/2\sigma^2} + e^{-(x+\mu)^2/2\sigma^2}\right]$ $x > 0;$	μ, σ	Normal	3.16	47
Kodlin's Distribution (AKODRN)	$(\eta+\gamma x)e^{-(\eta x + 1/2\,\gamma x^2)}$ $x > 0,$ $\gamma, \eta \geq 0$	γ, η	Exponential	3.17	49
Extreme Value Distributions (AMAXRN)	Maximum value: $\frac{1}{\sigma} \exp\left[-\frac{1}{\sigma}(x-\mu) - e^{-\frac{1}{\sigma}(x-\mu)}\right]$; $\sigma > 0$	μ, σ	Exponential	3.18	51
(AMINRN)	Minimum value: $\frac{1}{\sigma} \exp\left[\frac{1}{\sigma}(x-\mu) - e^{\frac{1}{\sigma}(x-\mu)}\right]$; $\sigma > 0$			3.16	51

TABLE 3.1 (Continued)

Name of Distribution (Function Table)	Functional Form	Parameters To Be Specified	Other Random Number Generators Used	Location of Algorithm to Generate Random Numbers	
				Subsection	Page
Weibull (WIBLRN)	$\frac{\eta}{\lambda}(x-\epsilon)^{\eta-1} e^{-\frac{(x-\epsilon)^{\eta}}{\lambda}}$ $x \geq \epsilon$ $\eta, \lambda > 0$	ϵ, η, λ	Exponential	3.19	53
Johnson System (SLRN)	S_L: $\frac{\eta}{\sqrt{2\pi}(x-\epsilon)} \cdot \exp\left\{-\frac{\eta^2}{2}\left[\frac{\gamma}{\eta} + \ln(x-\epsilon)\right]^2\right\}$ $x > 0$ $x \geq \epsilon$	ϵ, η, γ	Normal	3.20.1	55
(SBRN)	S_B: $\frac{\eta}{\sqrt{2\pi}} \frac{\lambda}{(x-\epsilon)(\lambda-x+\epsilon)} \cdot \exp\left\{-\frac{1}{2}\left[\gamma + \eta \ln\left(\frac{x-\epsilon}{\lambda-x+\epsilon}\right)\right]^2\right\}$ $\eta, \lambda > 0$ $\epsilon \leq x \leq \epsilon + \lambda$	$\eta, \gamma, \lambda, \epsilon$	Normal	3.20.2	57
(SURN)	S_U: $\frac{\eta}{\sqrt{2\pi}} \frac{1}{\sqrt{(x-\epsilon)^2 + \lambda^2}} \cdot \exp\left[-\frac{1}{2}\left(\gamma + \eta \ln\left\{\left(\frac{x-\epsilon}{\lambda}\right) + \left[\left(\frac{x-\epsilon}{\lambda}\right)^2 + 1\right]^{1/2}\right\}\right)^2\right]$ $\eta, \lambda > 0$	$\eta, \gamma, \lambda, \epsilon$	Normal	3.20.3	59
Pearson System (TYP1RN)	Type I: $C\left(1 + \frac{x}{a_1}\right)^{m_1}\left(1 - \frac{x}{a_2}\right)^{m_2}$ $m_1, m_2 > 1$ $a_1 < x < a_2$	a_1, a_2, m_1, m_2	Gamma	3.21.1	61
(TYP2RN)	Type II: $C\left(1 - \frac{x^2}{a^2}\right)^{m}$ $m > -1$ $-a < x < a$	a, m	Gamma	3.21.2	63
(TYP3RN)	Type III: $C\left(1 + \frac{x}{a}\right)^{\gamma a} e^{-\gamma x}$ $-a < x < a$	γ, a	Gamma	3.21.3	65

TABLE 3.1 (Continued)

Name of Distribution	Functional Form	Parameters To Be Specified	Other Random Number Generators Used	Location of Algorithm to Generate Random Numbers	
				Subsection	Page
(TYP4RN)	Type IV: $C\left(1+\frac{x^2}{a^2}\right)^{-m} e^{-\gamma \tan^{-1}(x/a)}$	m, γ, a	Uniform, Exponential	3.21.4	67
(TYP5RN)	Type V: $Cx^{-p} e^{-\gamma/x}$ $p > 1$; $\gamma, x > 0$ $\gamma, x < 0$	p, γ	Gamma	3.21.5	69
(TYP6RN)	Type VI: $C(x-a)^{q_2} x^{-q_1}$ $x - a > 0$ $q_1 > q_2 + 1 > 0$	a, q_1, q_2	Gamma	3.21.6	71
(TYP7RN)	Type VII: $C\left(1+\frac{x^2}{a^2}\right)^{-m}$; $m > 3.5$	a, m	Normal, Gamma	3.21.7	73
(TYP8RN)	Type VIII: $C\left(1+\frac{x}{a}\right)^{-m}$; $0 \le m \le 1$ $1 + x/a > 0$	a, m	Uniform	3.21.8	75
(TYP9RN)	Type IX: $C\left(1+\frac{x}{a}\right)^{m}$ $1 + x/a > 0$	a, m	Uniform	3.21.9	77
(TP10RN)	Type X: $\frac{1}{\sigma} e^{-x/\sigma}$; $\sigma > 0$ $x > 0$	σ	Exponential	3.21.10	79
(TP11RN)	Type XI: $C(b/x)^m$; $x > b$ $m > 1$	m, b	Uniform	3.21.11	81
(TP12RN)	Type XII: $C\left\{\frac{\sigma\left(\sqrt{3+\beta_1}+\sqrt{\beta_1}\right)+x}{\sigma\left(\sqrt{3+\beta_1}-\sqrt{\beta_1}\right)-x}\right\}^{\sqrt{\beta_1/(3+\beta_1)}}$ $\beta_1 > 0$ $-\sigma\sqrt{3+\beta_1}+\sqrt{\beta_1} < x < \sigma\sqrt{3+\beta_1}-\sqrt{\beta_1}$	β_1, σ	Beta	3.21.12	83
Histogram (AHSTRN)	Not applicable	Upper and lower limits and intermediate break points in distribution	Uniform	3.22	86

argument. In the flow diagrams, these are indicated as U(0, 1), E(0, 1) and N(0, 1), respectively.

3.1 UNIFORM RANDOM NUMBER GENERATORS

The uniform random number generator is, of course, fundamental to all random number generation. For the purposes here, it is assumed that the computer system available will have such a generator as part of the basic software package. If one is not available or the generator is expected to be faulty, the machine independent package presented in Appendix B (MIRAN) can be used. The following paragraphs describe the technique used in most computers for generating random numbers and provide insight into the assessment of such generators.

The method used for almost all uniform generators is the multiplicative congruential method.[7] A sequence of integers, $x_0, x_1, \ldots$, is generated by the congruence

$$x_{n+1} = x_n \cdot \lambda (\text{mod } 2^P) \quad .$$

Here P is the number of bits (excluding sign) in a word on the particular computer employed and λ is called the generator which is a carefully selected integer as described below. From this sequence random fractions are produced using

$$R_n = x_n \cdot 2^{-P} \quad .$$

The sequence of random fractions, $R_1, R_2, \ldots$, is output by the subroutine in floating point form.

On most computers the multiplicative congruential method is accomplished by an integer multiplication of x_n and λ. Only the low-order half (P bits) of the product is retained as x_{n+1}. This is then treated as a binary fraction, converted to floating point, and normalized.

This method is fast and will produce numbers whose properties approximate randomness sufficiently close for valid use in Monte Carlo simulations provided the following caveats are observed.

1. Choose a generator, λ, with particular care. In particular, generators with a small number of '1' bits in their binary representation should be avoided. A number of generators of the form $2^{18} \pm 3$, $2^{24} \pm 3$, $2^{16} \pm 3$, etc., are particularly abundant. At one time, they were used because they were thought to be good and especially fast. However, further research has shown them to be faulty and a number of simulations have produced erroneous results as a consequence. Small generators such as $\lambda = 101$ are also faulty and must be avoided. The generators $\lambda = 5^{15}$ or $\lambda = 5^{13}$ have been well tested and are quite safe to use.[1]

2. Check the computer word length. It is best for P to be at least 35 in the congruence. For machines with $P \leq 32$ a multiple precision multiplication should be used to generate an adequate congruence.

3. Do not trust, on blind faith, random number routines distributed by the computer manufacturers with standard subroutine libraries. These have been found to contain, with high probability, the faulty generator values.

The uniform random number generator will be referred to as UNFRN(R) in subsequent routines and U(0, 1) in the flow diagrams.

3.2 EXPONENTIAL DISTRIBUTION

The simplest method to generate random numbers from the exponential distribution, $f(x) = e^{-x}$, is to use the inverse solution,

$$x = -\ln(R_u) \quad ,$$

where R_u is a uniform random number. This is not, however, the fastest method. An extremely rapid technique has been developed by G. Marsaglia[3] which, although it is several times faster than the logarithm, requires a sizable block of computer storage (~600 words). When computer storage is critical or when the exponential distribution is not of crucial importance, the Von Neumann rejection technique is a good general method. This method, usually faster than the logarithm, is shown in Fig. 3-1.

To select from a generalized exponential, $(1/\lambda)e^{-[(x-\epsilon)/\lambda]}$, it is merely necessary to select from e^{-x} then multiply by λ and add ϵ. For best efficiency in general, the basic exponential subroutine should select from e^{-x}, and it should be left up to the calling program to supply the multiplication and addition where needed.

The exponential distribution is referred to as EXPRN(R) in subsequent routines and as E(0,1) in the flow diagrams.

Sample Routines

Simplest method (use inline in calling program):

```
R = -ALOG (UNFRN(R))
```

Von Neumann rejection technique:

```
      FUNCTION EXPRN(DUMMY)
      I = 0
100   X = UNFRN(X)
105   Y = UNFRN(X)
      IF (X. LT. Y) GO TO 120
110   X = UNFRN(X)
      IF (X. LT. Y) GO TO 105
115   I = I+1
      GO TO 100
120   EXPRN = X+I
      RETURN
      END
```

$$f(x) = e^{-x} \quad ; \quad x \cdot 0$$

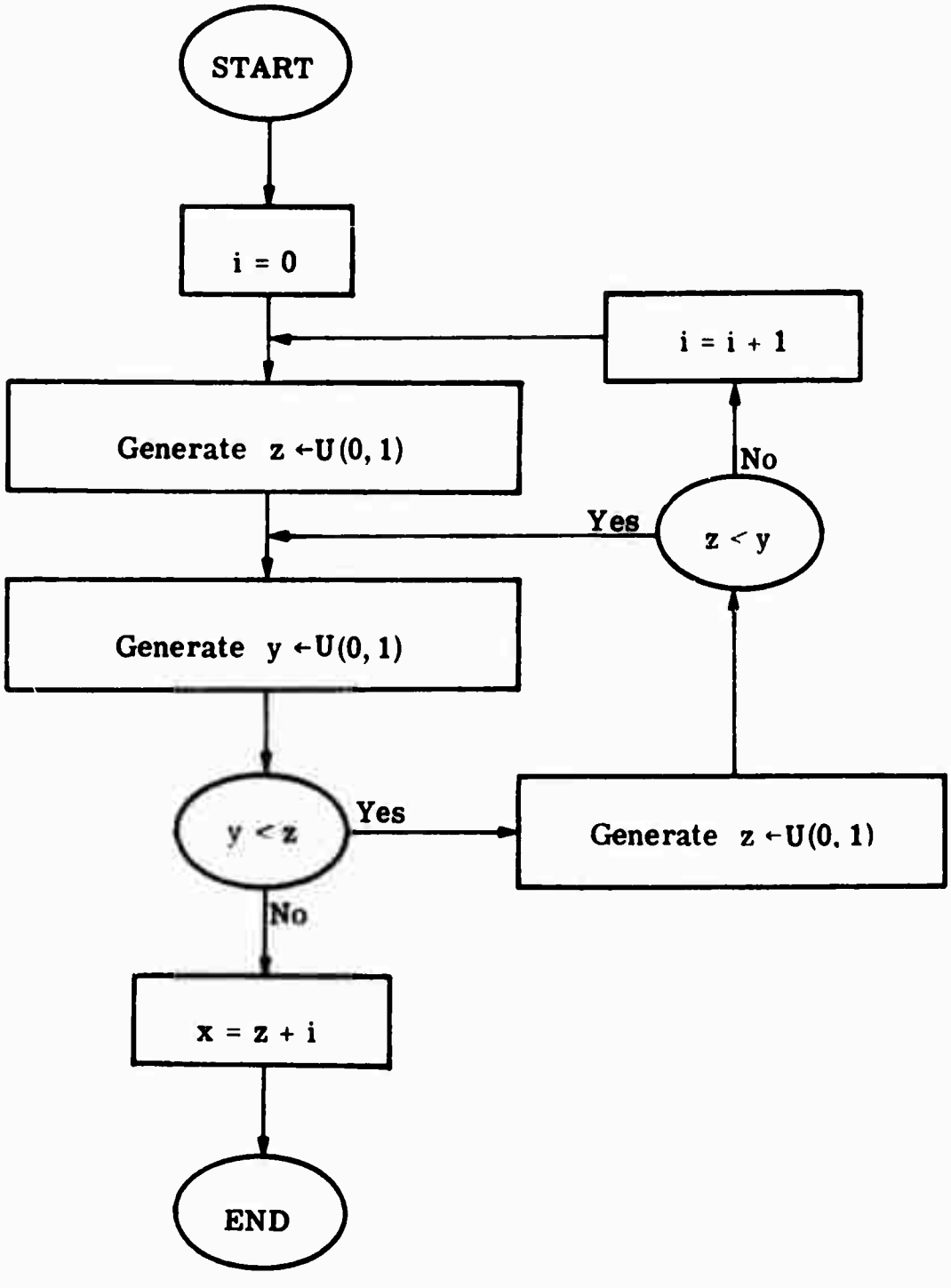


Figure 3-1. Random number generation algorithm for exponential distribution

3.3 NORMAL DISTRIBUTION

The normal distribution, $f(x) = 1/(\sigma\sqrt{2\pi})e^{-(x-\mu)^2/2\sigma^2}$, has received considerable attention by the designers of random number generators. One of the earliest methods, which is still found frequently in simulations today, uses the central limit theorem to approximate the normal by summing up several uniform random variables.[6] This approach has two serious defects. First, it is only an approximation. Second, it is much slower than other methods. The fastest method by far is a technique designed by G. Marsaglia.[5] However, considerable storage is needed for this technique. Another technique by Marsaglia,[4] illustrated in Fig. 3-2, is fairly fast without requiring much computer storage. This is the best technique known for general usage.

As with the exponential routine, the basic normal random number generator should be written to select from the normal distribution with unit mean and zero standard deviation (referred to as ANRMRN in the routines and as N(0, 1) in the flow diagrams). It is then left up to the calling program to multiply by the standard deviation and add the mean if a generalized normal deviate is required. That is, for a distribution with mean μ and variance σ^2, the correct random number would be $\sigma N(0,1) + \mu$, where $N(0,1)$ is a random number from a distribution with $\mu = 0$ and $\sigma^2 = 1$.

Sample Routine

```
      FUNCTION ANRMRN (DUMMY)
      R = UNFRN(R)
      IF (R. GT. 0. 8638) GO TO 10
      ANRMRN = 2. *(UNFRN(X) + UNFRN(Y) + UNFRN(Z) - 1. 5)
      RETURN
10    IF (R. GT. 0. 9745) GO TO 20
      ANRMRN = 1. 5*(UNFRN(X) + UNFRN(Y) - 1. 0)
      RETURN
```

```
20      IF (R. GT. 0. 997302039) GO TO 100
25      X = 6. *UNFRN(X) - 3. 0
        Y = 0. 358*UNFRN(X)
        XSQ = X*X
        GX = 17. 49731196*EXP(-XSQ*. 5)
        AX = ABS(X)
        IF (AX. GT. 1. 0) GO TO 30
        IF (Y. GT. (GX-17. 44392294 + 4. 73570326*XSQ + 2. 15787544*AX))
          GO TO 25
        ANRMRN = X
        RETURN
30      AX3 = 2. 36785163*(3-AX)**2
        IF (AX. GT. 1. 5) GO TO 40
        IF (Y. GT. (GX-AX3-2. 15787544*(1. 5-AX))) GO TO 25
        ANRMRN = X
        RETURN
40      IF (Y. GT. (GX-AX3)) GO TO 25
        ANRMRN = X
        RETURN
100     X = SQRT (9+2*EXPRN(X))
        IF (UNFRN(X). GT. 3/X) GO TO 100
        IF (UNFRN(X). GT. 0. 5) X = -X
        ANRMRN = X
        RETURN
        END
```

$$f(x) = (2\pi)^{-1/2} e^{-x^2/2} \quad ; \quad -\infty < x < \infty$$

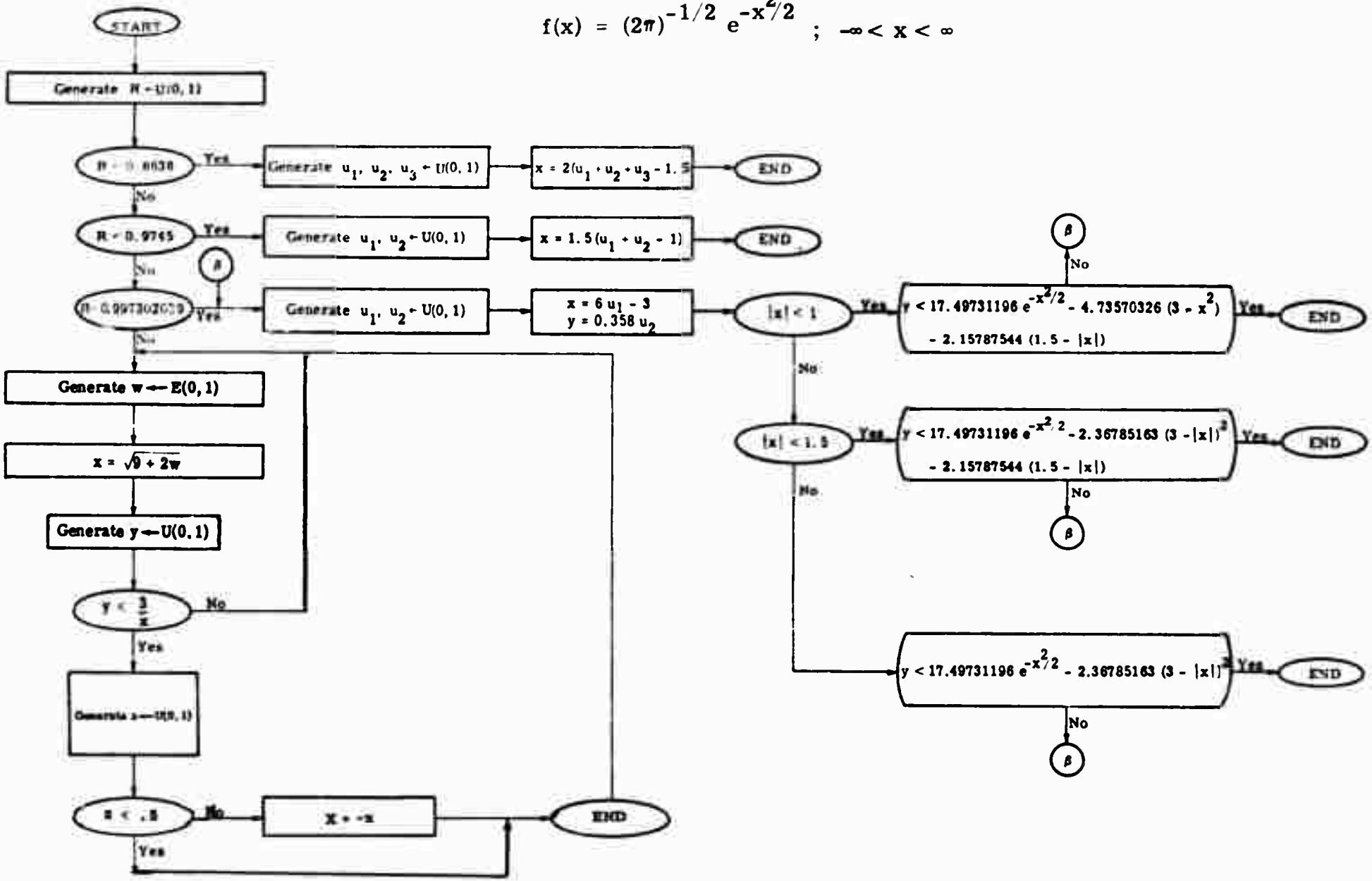


Figure 3-2. Normal distribution

3.4 THE BINOMIAL DISTRIBUTION

The binomial distribution, $p_k = \binom{n}{k} p^k (1-p)^{n-k}$, is a discrete distribution describing the number of successes encountered in a series of Bernoulli trials. It has two parameters, p, the probability of success in a single trial, and n, the number of trials in the series.

The algorithm for selection from the binomial distribution is divided into three subranges for the parameter p. For moderate values of p, the random number generation is based on a straightforward simulation of the underlying basis for the distribution; n Bernoulli trials are generated and the number of successes are counted. For small values of p, it becomes more efficient to use a technique based on the geometric distribution. Conversely, for large values of p it is efficient to reverse the geometric technique and perform the counting on the number of failures rather than successes.

For large values of n, all three algorithms become inefficient; the computing time involved is directly proportional to n. The binomial distribution approximates a normal distribution with mean np and standard deviation $\sqrt{np(1-p)}$ for large n. One should consider replacing the binomial with the approximate normal for large values of n ($n > 10\ p/(1-p)$ or $n > 10\ (1-p)/p$).

Sample Subroutines

For $p < .25$

```
      FUNCTION KBINOM (N, ALNQ)
C     ALNQ IS -ALOG (1. -P)
      KBINOM = 0
      M = 0
5     R = EXPRN(R)
      J = 1 + R/ALNQ
      M = M + J
      IF (M - N)10, 15, 20
10    KBINOM = KBINOM + 1
      GO TO 5
15    KBINOM = KBINOM + 1
20    RETURN
      END
```

For $.25 \leq p \leq .75$

```
      FUNCTION KBINOM (N, P)
      KBINOM = 0
      DO 15 M = 1, N
      R = UNFRN (R)
      IF (R. LT. P) KBINOM = KBINOM + 1
15    CONTINUE
      RETURN
      END
```

For $p > .75$

```
      FUNCTION KBINOM (N, ALNP)
C     ALNP IS -ALOG (P)
      KBINOM = N
      M = 0
5     R = EXPRN (R)
      J = 1 + R/ALNP
      M = M + J
      IF (M-N)10, 15, 20
10    KBINOM = KBINOM - 1
      GO TO 5
15    KBINOM = KBINOM -1
20    RETURN
      END
```

$$P_k = \binom{n}{k} p^k (1-p)^{n-k} \quad ; \qquad \text{For } p < 0.25$$

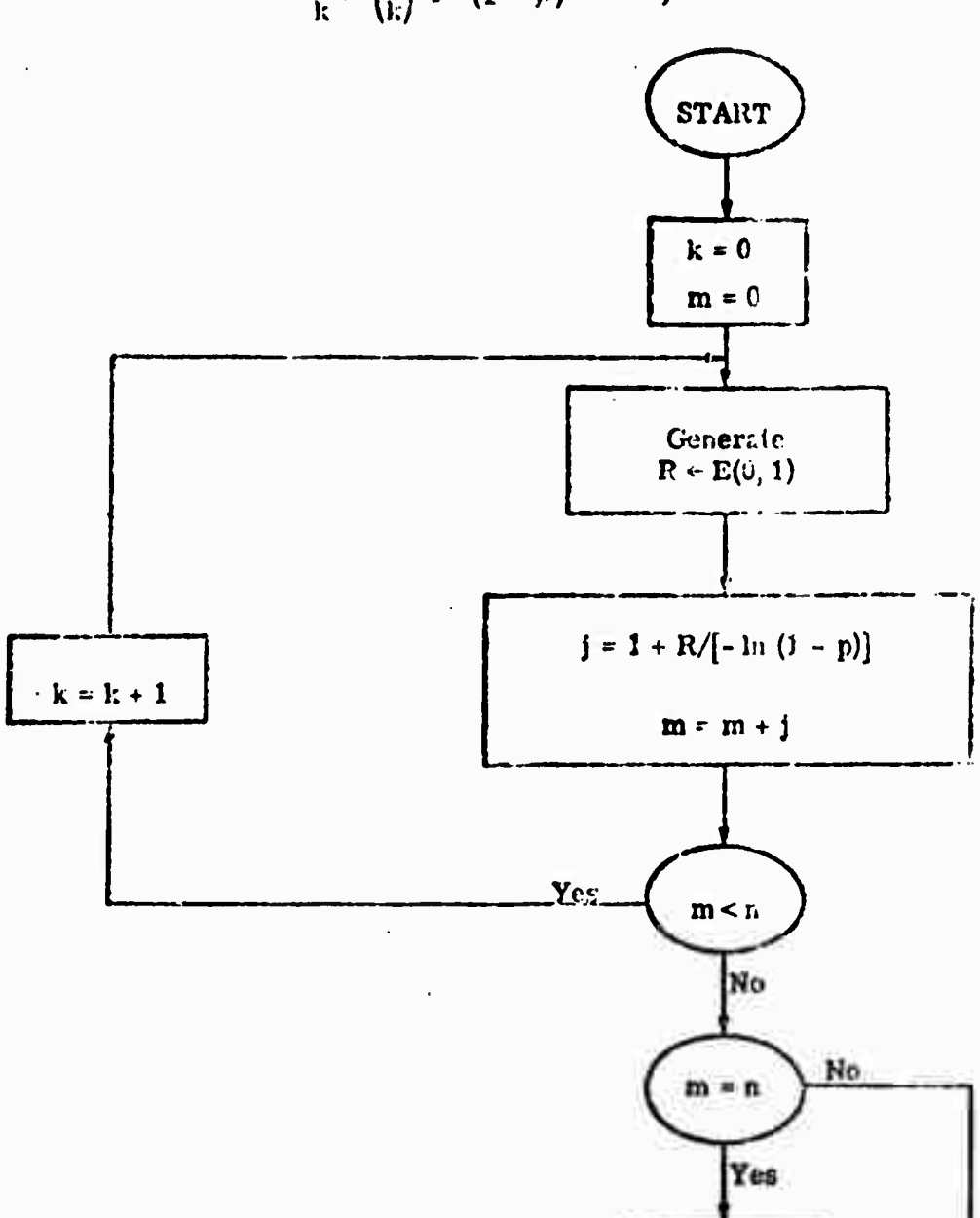


Figure 3-3. Random number generation algorithm for binomial distribution

$$p_k = \binom{n}{k} p^k (1-p)^{n-k} \quad ; \qquad \text{For } 0.25 \le p \le 0.75$$

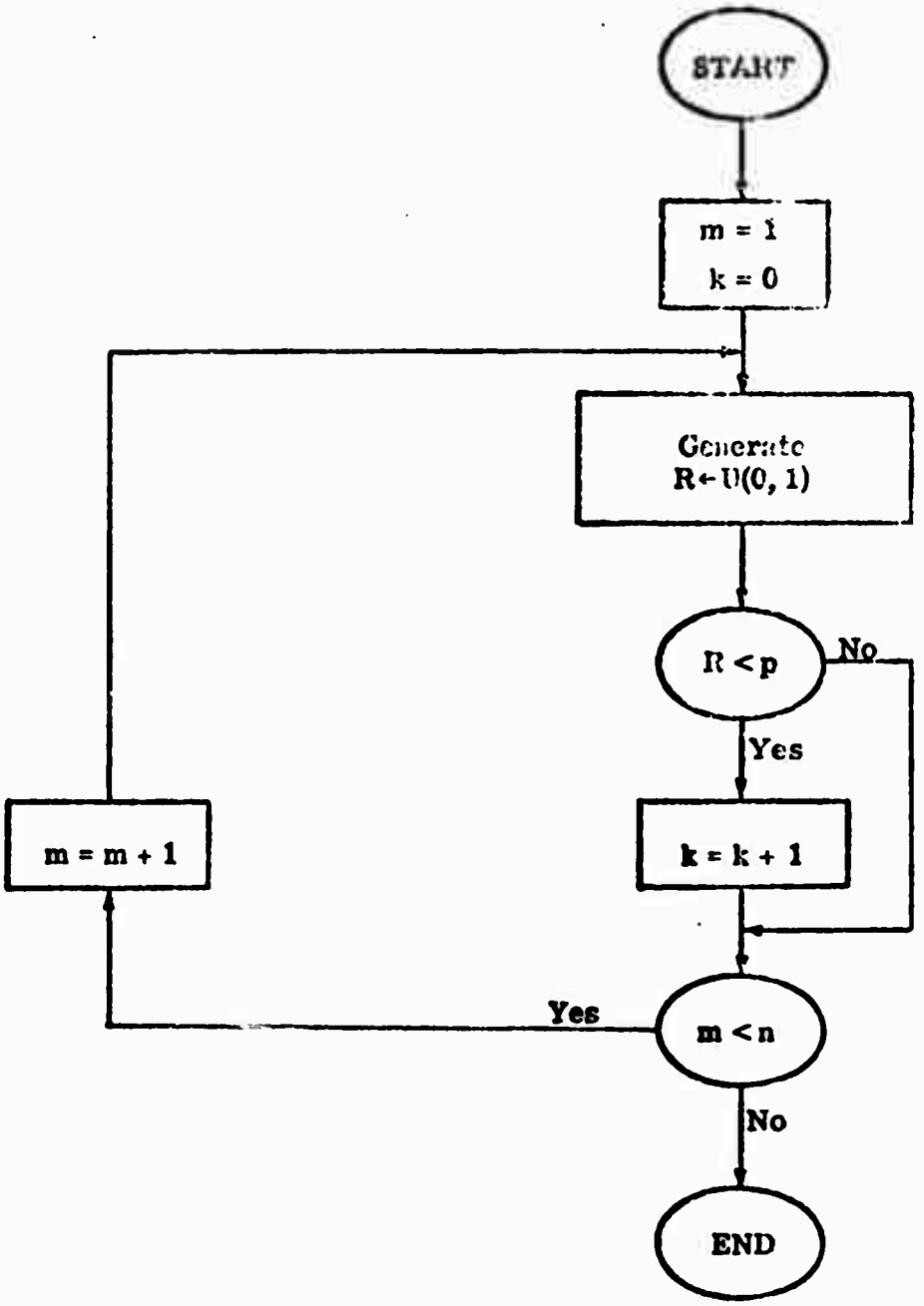


Figure 3-4. Random number generation algorithm for binomial distribution (continued)

$$p_k = \binom{n}{k} p^k (1-p)^{n-k} \quad ; \quad \text{For } p > 0.75$$

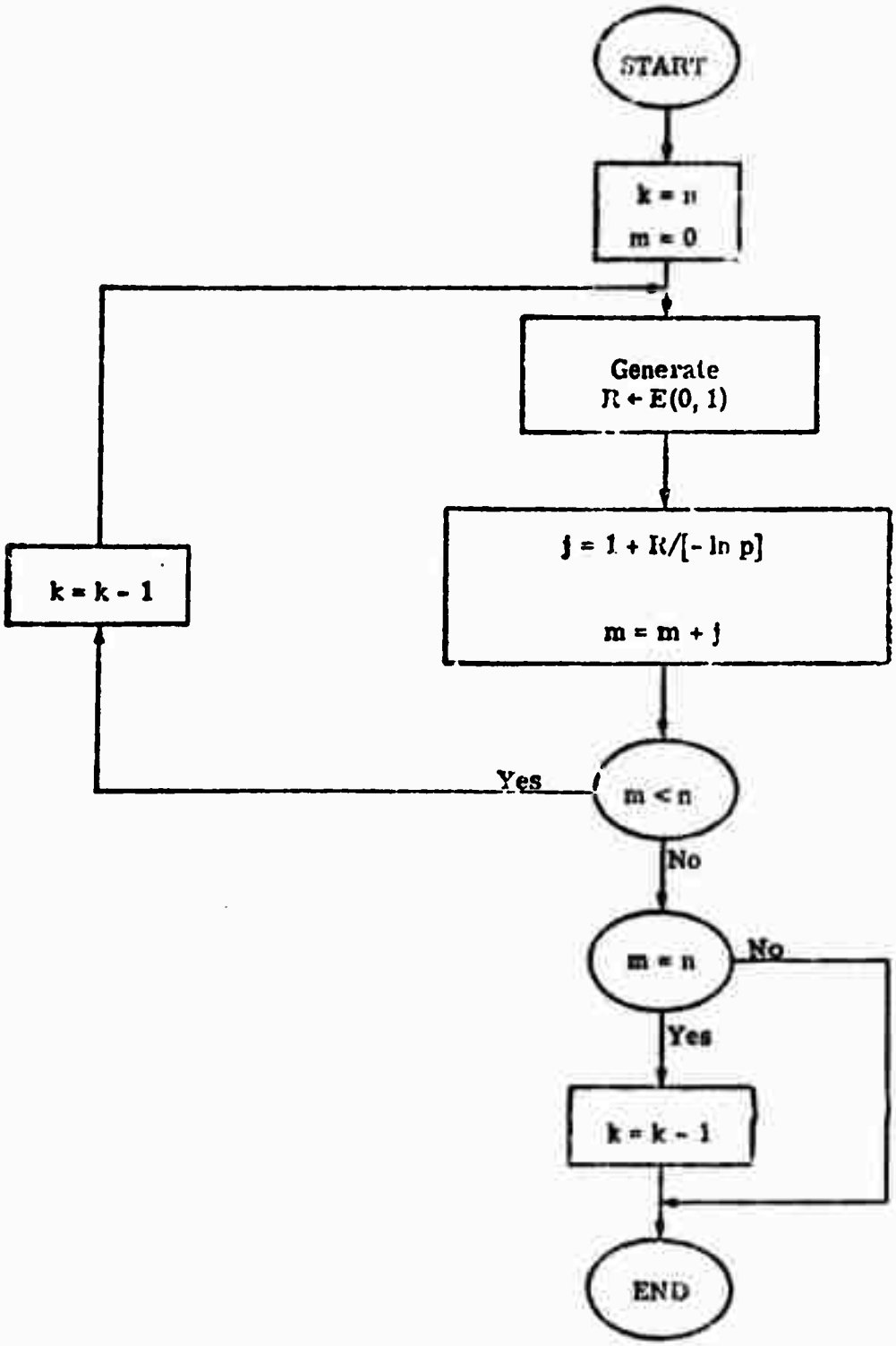


Figure 3-5. Random number generation algorithm for binomial distribution (continued)

3.5 THE MULTINOMIAL DISTRIBUTION

The multinomial distribution,

$$p(k_1, k_2, \ldots, k_m) = \binom{n}{k_1 k_2 \ldots k_m} p_1^{k_1} p_2^{k_2} \cdots\cdots p_m^{k_m}$$

is a generalization of the binomial distribution to trials having m different outcomes with discrete probabilities. Random number generation is accomplished by a straightforward simulation of the underlying process of identical trials. Note that a 'random number' for this distribution is an array containing the number of realizations of each possible outcome.

Sample Routine

```
      SUBROUTINE MULNOM (N, M, K, P)
      DIMENSION K(M), P(M)
C     P IS INPUT ARRAY OF PROBABILITIES
C     K IS OUTPUT ARRAY OF OUTCOMES
      DO 10 J = 1, M
10    K(J) = 0
      DO 30 I = 1, N
      R = UNFRN (R)
      DO 20 J = 1, M
      R = R - P(J)
      IF (R. LT. 0.) GO TO 30
20    CONTINUE
30    K(J) = K(J) + 1
      RETURN
      END
```

$$p(k_1, k_2, \ldots, k_m) = \binom{n}{k_1\ k_2 \ldots\ k_m} p_1^{k_1}\ p_2^{k_2} \ldots\ p_m^{k_m}$$

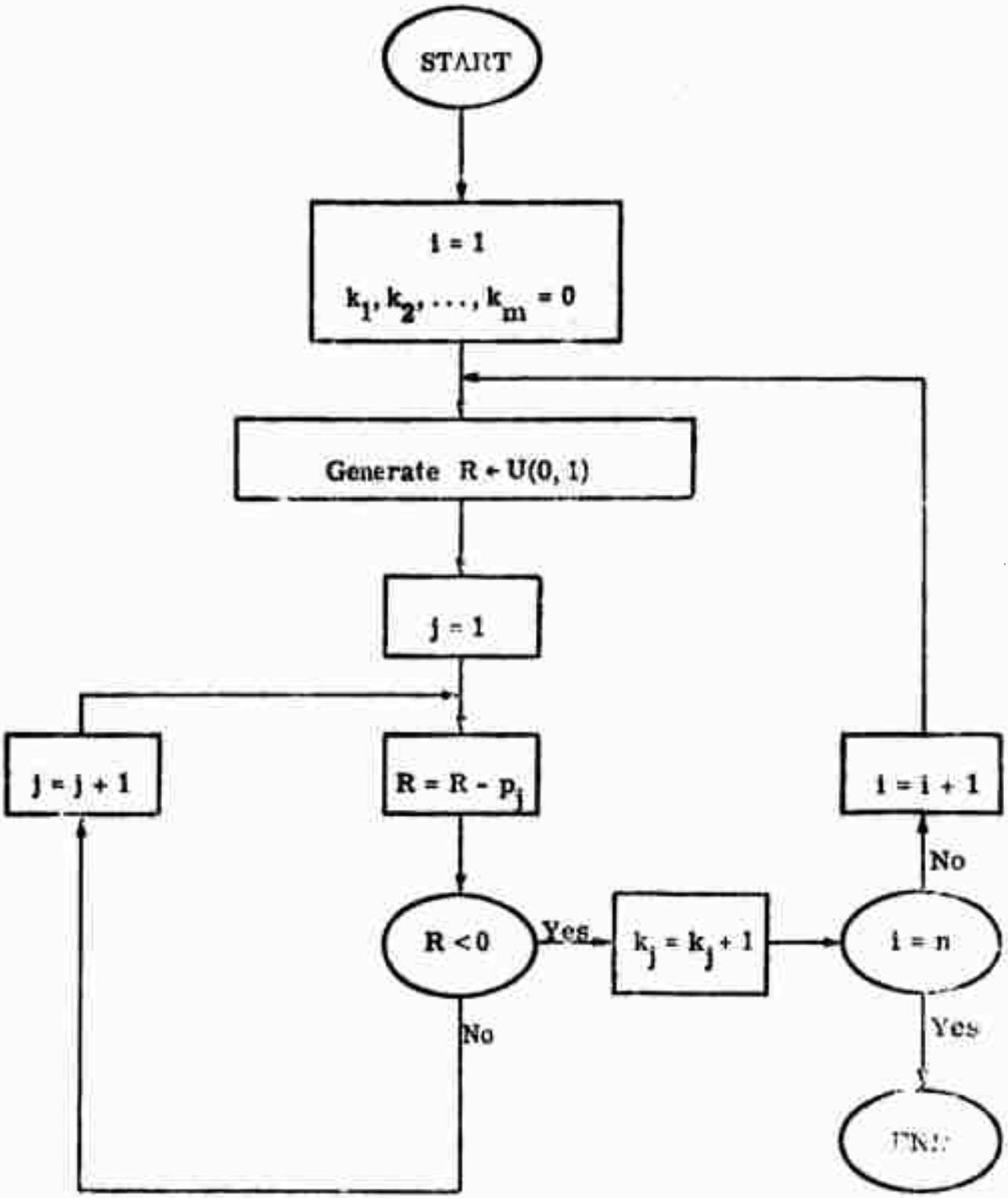


Figure 3-6. Random number generation algorithm for multinomial distribution

3.6 POISSON DISTRIBUTION

The Poisson distribution, $p_k = e^{-\lambda}\frac{\lambda^k}{k!}$, is a discrete distribution describing the number of occurrences in an interval when the rate of occurrence is a constant. The technique for selecting from the Poisson distribution is a combination-transformation method described in Ref. 2.

The computer time spent in this selection is directly proportional to λ, the mean value of the Poisson variable. For large λ, this selection can be very time consuming. It is possible to approximate the Poisson distribution by a normal distribution with a mean of λ and a standard deviation of $\sqrt{\lambda}$ for λ sufficiently large ($\lambda > 10$).

Sample Routine

```
      FUNCTION KPOIS (EXPLAM)
C     EXPLAM IS EXP (-LAMBDA)
      Y = 1.0
      KPOIS = 0
5     Y = Y * UNFRN (Y)
      IF (Y. GT. EXPLAM) GO TO 10
      KPOIS = KPOIS + 1
      GO TO 5
10    RETURN
      END
```

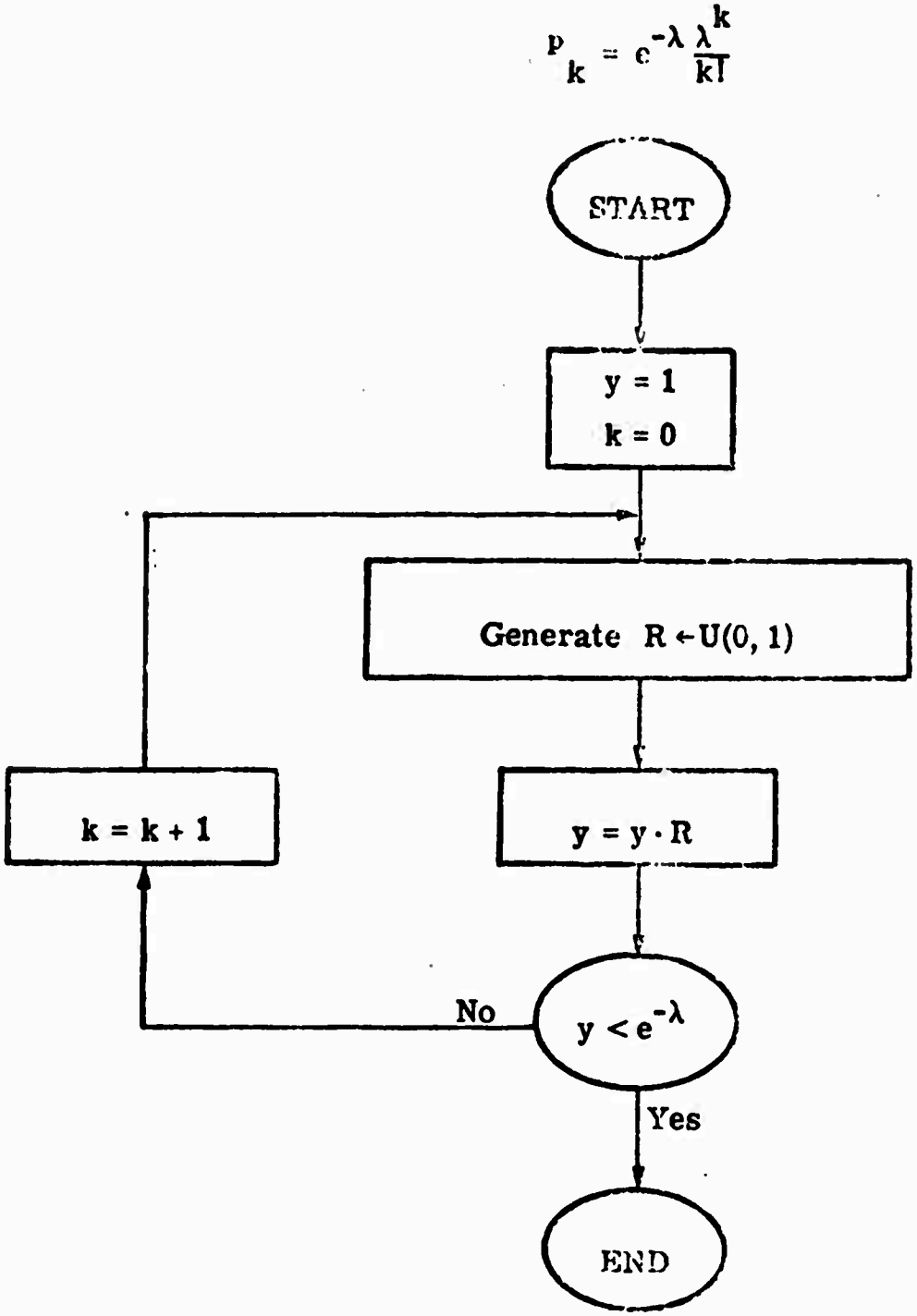


Figure 3-7. Random number generation algorithm for Poisson distribution

3.7 HYPERGEOMETRIC DISTRIBUTION

The hypergeometric distribution,

$$p_k = \frac{\binom{M}{k}\binom{M-N}{n-k}}{\binom{N}{n}}$$

describes sampling without replacement. It has the parameters N, the size of the total population, n, the size of the population sampled, and M, the number of events in the total population. The random variable k is the number of events occurring in the sample. The hypergeometric distribution is generated by simulating sampling without replacement.

Sample Routine

```
      FUNCTION KHYPRG (NTOT, MTOT, N)
C     NTOT IS TOTAL POPULATION SIZE, MTOT IS TOTAL
C     EVENTS IN POPULATION, N IS SAMPLE SIZE
      KHYPRG = 0
      EM = MTOT
      EN = NTOT
      DO 10 I = 1, N
      P = EM/EN
      R = UNFRN (R)
      IF (R. GT. P) GO TO 10
      KHYPRG = KHYPRG + 1
      EM = EM - 1.
10    EN = EN - 1.
      RETURN
      END
```

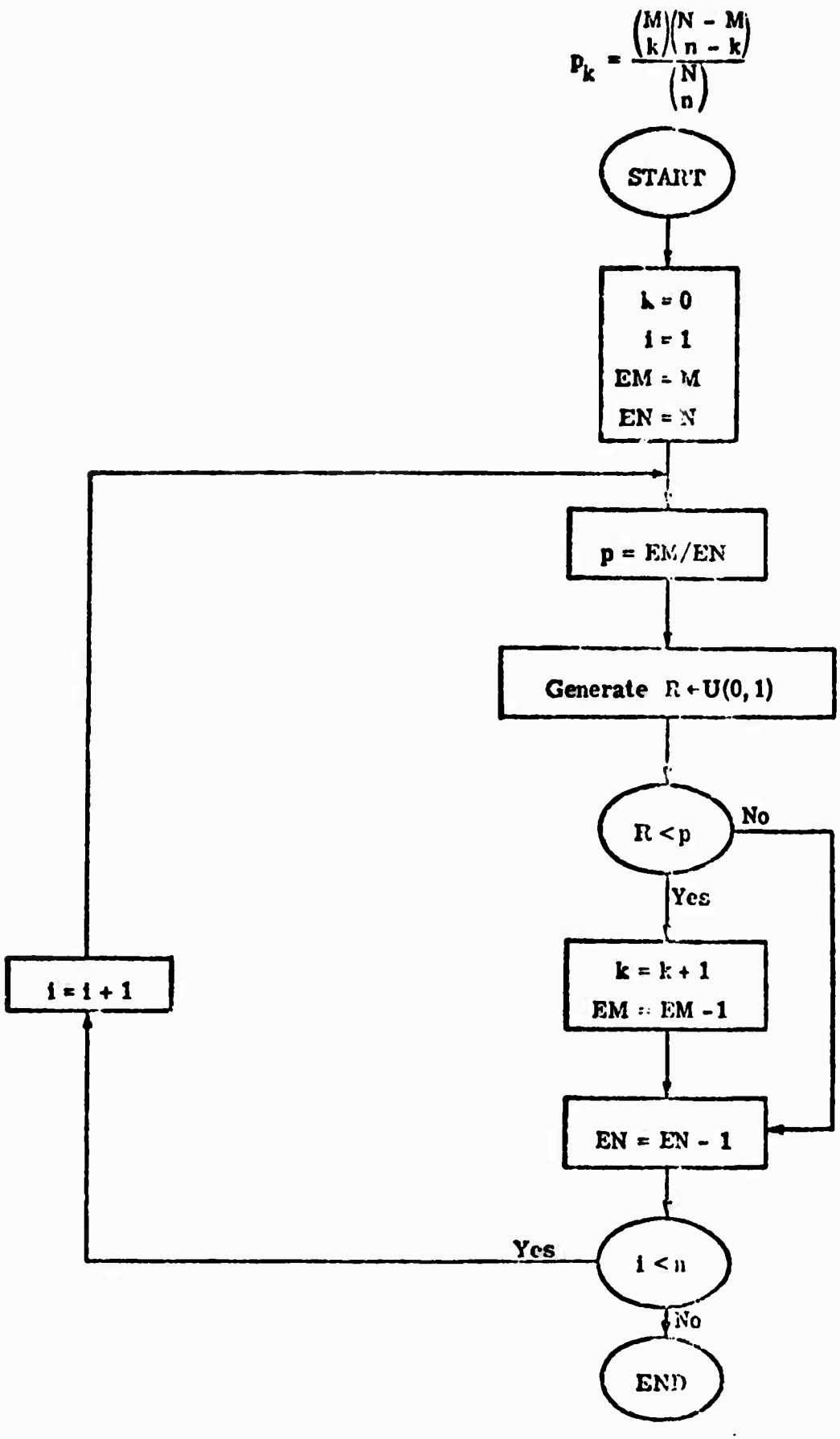


Figure 3-8. Random number generation algorithm for hypergeometric distribution

3.8 GEOMETRIC DISTRIBUTION

The geometric distribution, $p_k = p(1-p)^{k-1}$, describes the number of trials to the first success in a series of Bernoulli trials. For $p \geq .25$, the geometric distribution is most efficiently sampled by a direct solution of the discrete inverse equation. When $p < .25$, it becomes more efficient to generate a geometric variate by truncating an exponential random number.

Sample Routines

For $p < .25$:

```
      FUNCTION KGEOM (ALNQ)
C     ALNQ IS -ALOG (1 - P)
      R = EXPRN (R)
      KGEOM = 1 + R/ALNQ
      RETURN
      END
```

For $p \geq .25$:

```
      FUNCTION KGEOM (P)
      A = P
      Q = 1 - P
      KGEOM = 1
      R = UNFRN (R)
10    R = R - A
      IF (R. LT. 0) RETURN
      KGEOM = KGEOM + 1
      A = A * Q
      GO TO 10
      END
```

$$p_k = p\,(1 - p)^{k-1} \quad : \quad p < 0.25$$

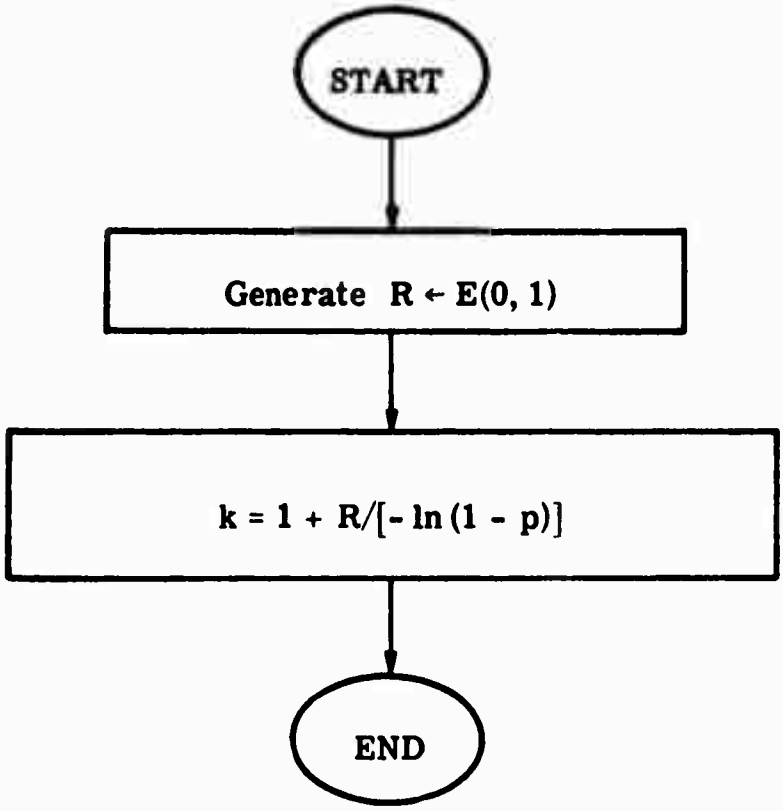


Figure 3-9. Random number generation algorithm for geometric distribution

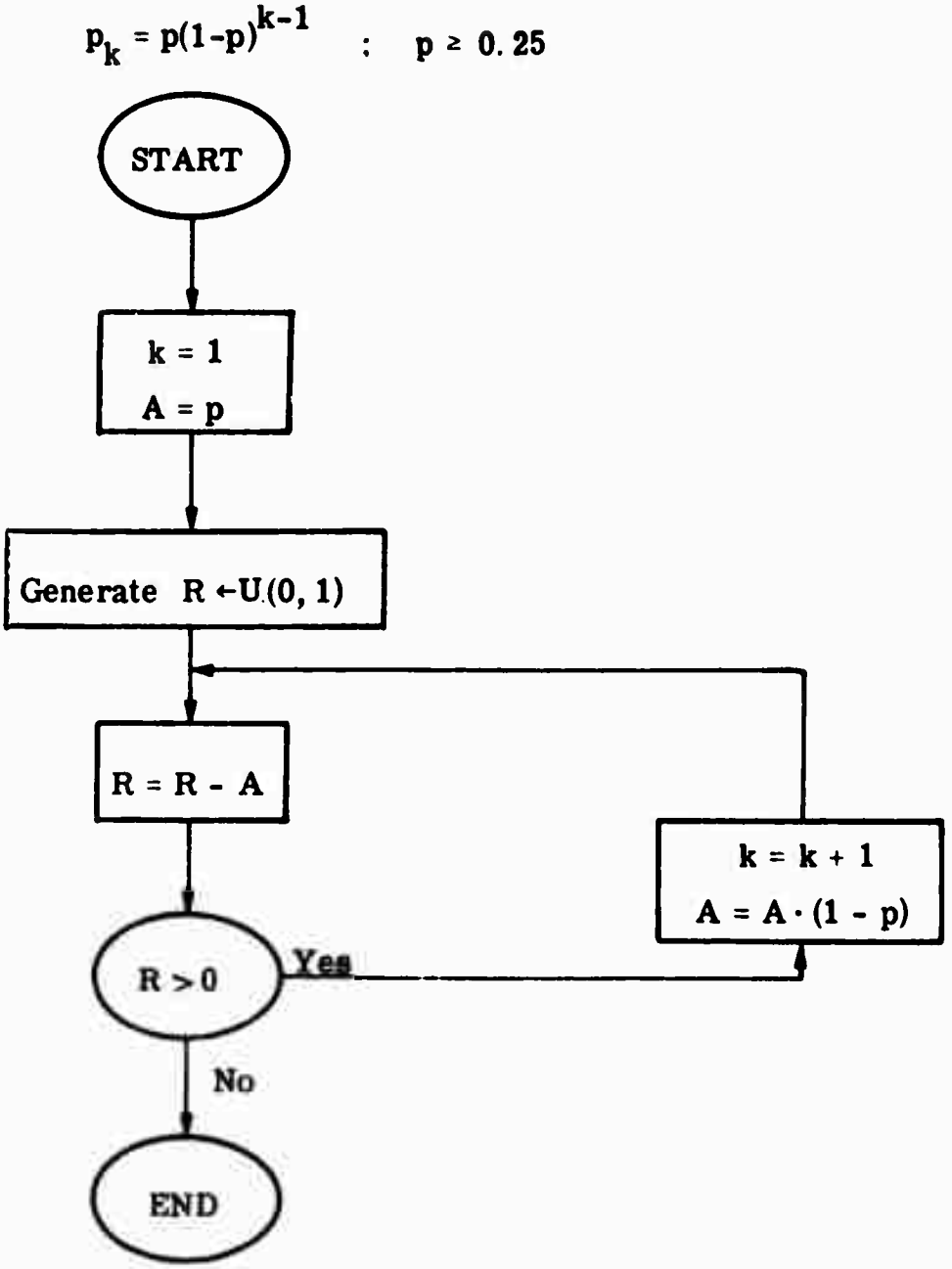


Figure 3-10. Random number generation algorithm for geometric distribution (continued)

3.9 PASCAL OR NEGATIVE BINOMIAL DISTRIBUTION

The Pascal distribution,

$$p_k = \binom{n+k-1}{k} (1-p)^n p^k ,$$

describes the number of successes occurring before the nth failure in a series of Bernoulli trials. For low or moderate values of p , the Pascal distribution is efficiently generated by a direct simulation of a sequence of Bernoulli trials. As p becomes large ($p > .75$), it becomes more efficient to sample by generating a geometric variate for the number of trials to each of the n failures.

Sample Routines

For $p \leq .75$:

```
      FUNCTION  KPASCL (P, N)
      KPASCL = 0
      DO 20 J = 1, N
10    R = UNFRN (R)
      IF (R. GT. P) GO TO 20
      KPASCL = KPASCL + 1
      GO TO 10
20    CONTINUE
      RETURN
      END
```

For $p > .75$:

```
      FUNCTION KPASCL (ALNP, N)
C     ALNP IS -ALOG(P)
      KPASCL = 0
      DO 10 J = 1, N
      I = EXPRN(R)/ALNP
10    KPASCL = KPASCL + I
      RETURN
      END
```

$$p_k = \binom{n+k-1}{k} (1-p)^n p^k \quad : \quad p \leq 0.75$$

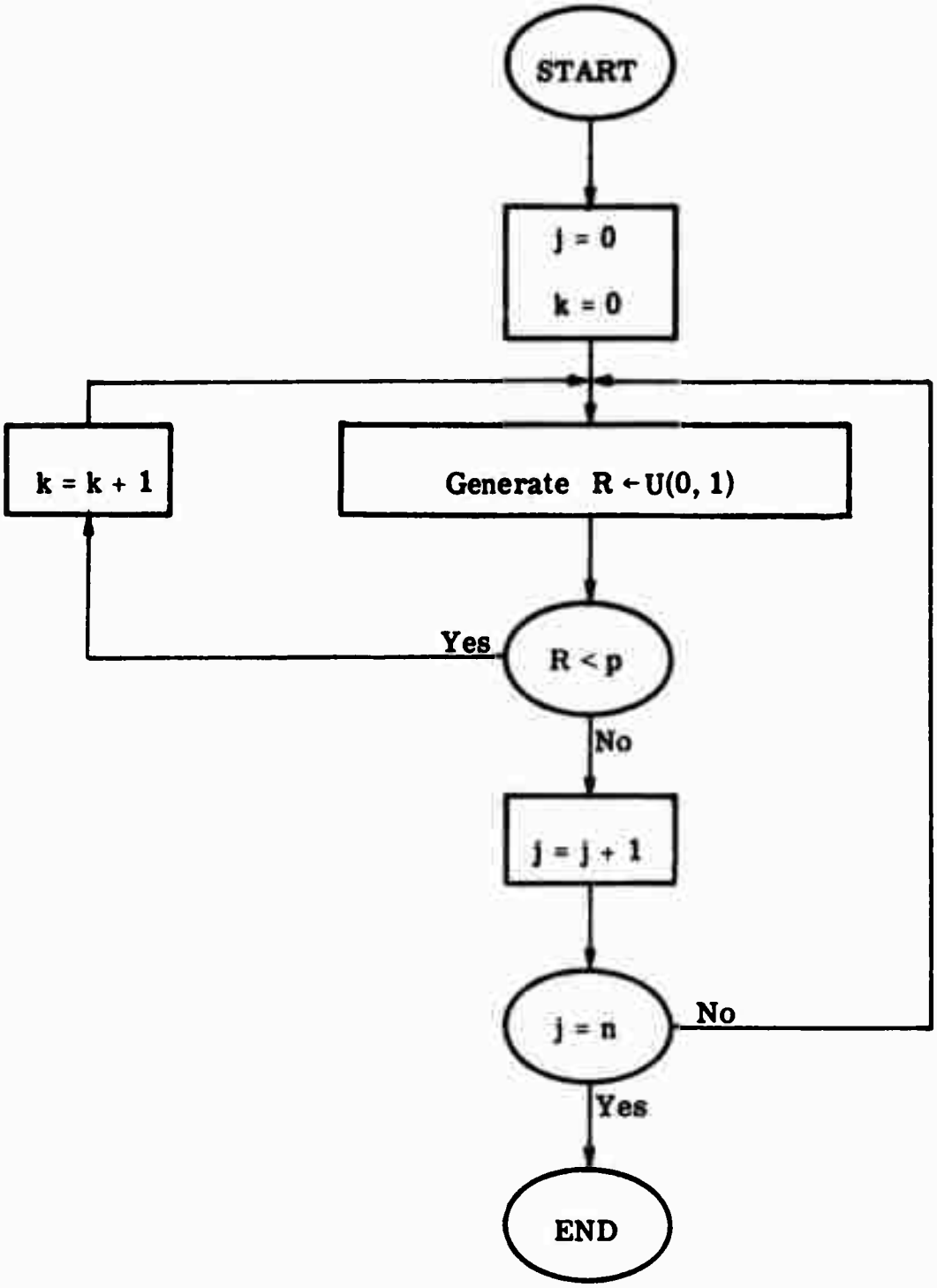


Figure 3-11. Random number generation algorithm for Pascal distribution

$$p_k = \binom{n+k-1}{k} (1-p)^n p^k \quad ; \quad p > 0.75$$

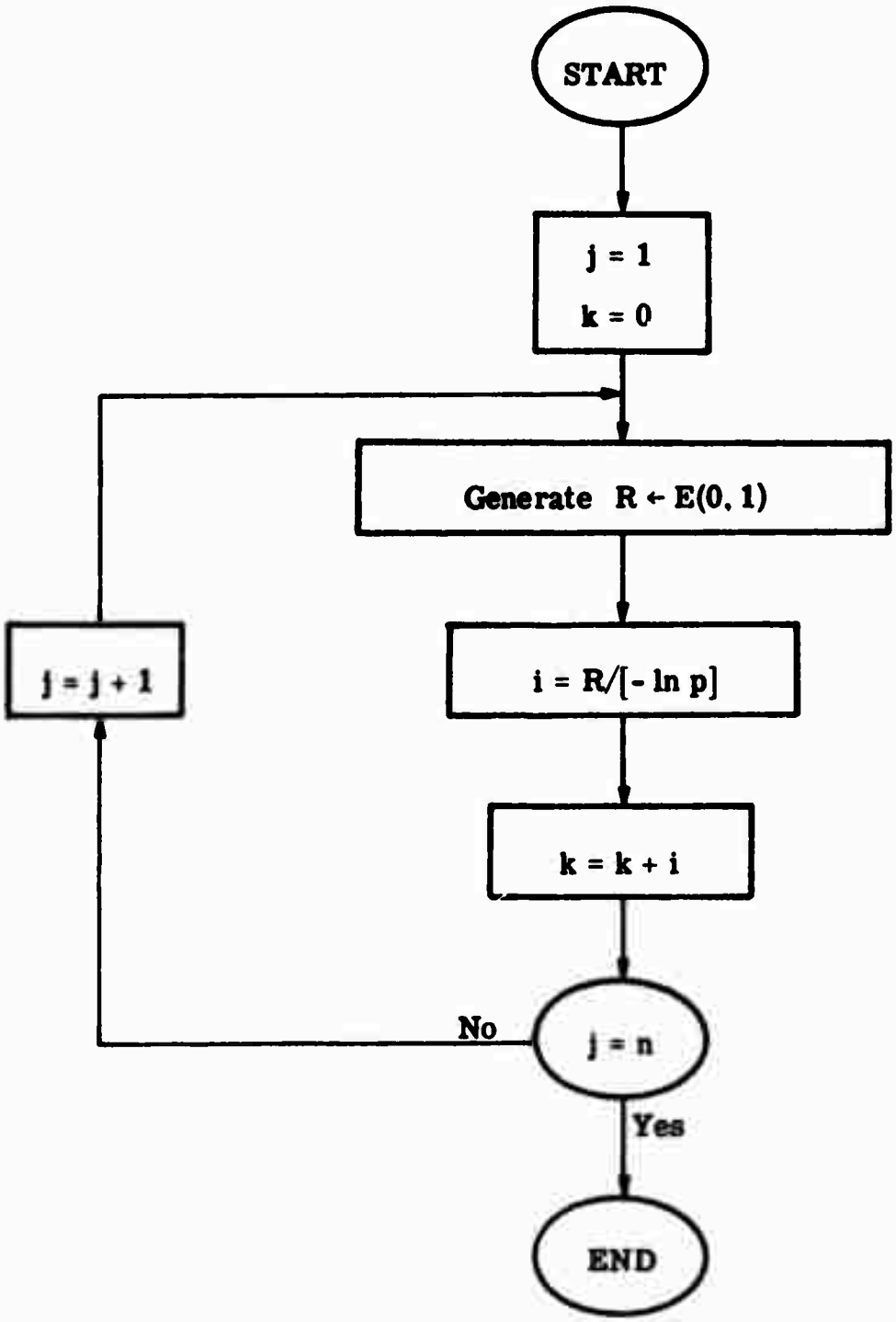


Figure 3-12. Random number generation algorithm for Pascal distribution (continued)

3.10 CAUCHY DISTRIBUTION

The Cauchy distribution,

$$f(x) = \frac{1}{\pi[1 + (x-\mu)^2]}, \quad -\infty < x < \infty$$

represents the distribution of the ratio of two normally distributed numbers. It also represents the tangent of a random angle. It is easily generated by a rejection technique which selects x and y uniformly in a unit circle, then calculates the tangent x/y.

Caution: The moments of the Cauchy distribution are infinite; the behavior of Cauchy variates in a simulation will be erratic.

Sample program:

```
      FUNCTION COCHRN (AMU)
10    X = UNFRN(Y)
      Y = 2. *UNFRN (X) - 1.
      IF (X * X + Y * Y. GT. 1) GO TO 10
      COCHRN = AMU + Y/X
      RETURN
      END
```

$$f(x) = \frac{1}{\pi\left[1 + (x - \mu)^2\right]}$$

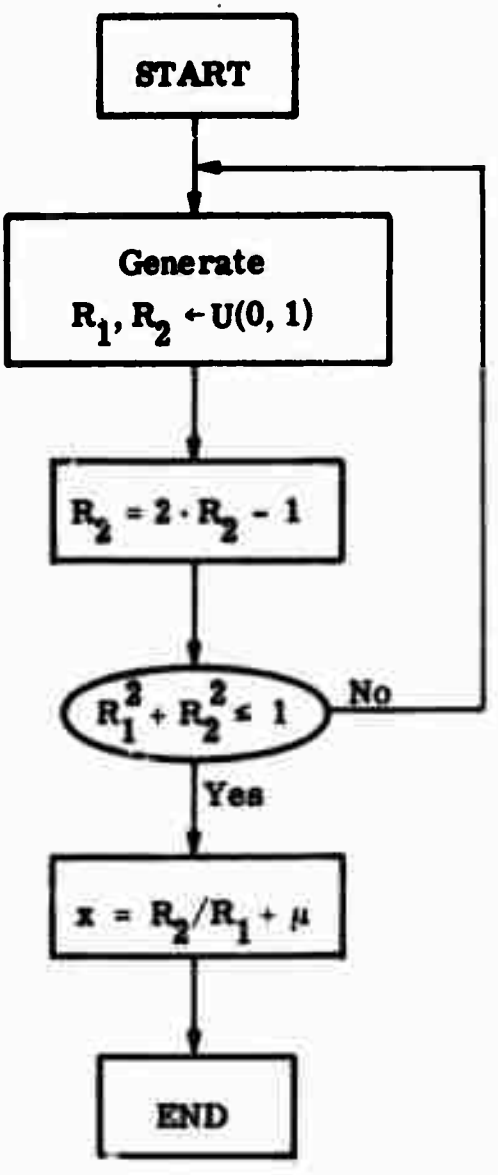


Figure 3-13. Random number generation algorithm for Cauchy distribution

3.11 RAYLEIGH DISTRIBUTION

The Rayleigh distribution,

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2} \quad ,$$

is derived as the radial error when the x and y errors are independent normal variates. It has a simple inverse which provides the most efficient method for generating Rayleigh variates.

Sample routine:

```
FUNCTION RAYLRN (SIGMA)
RAYLRN = SIGMA * SQRT (2.*EXPRN(R))
RETURN
END
```

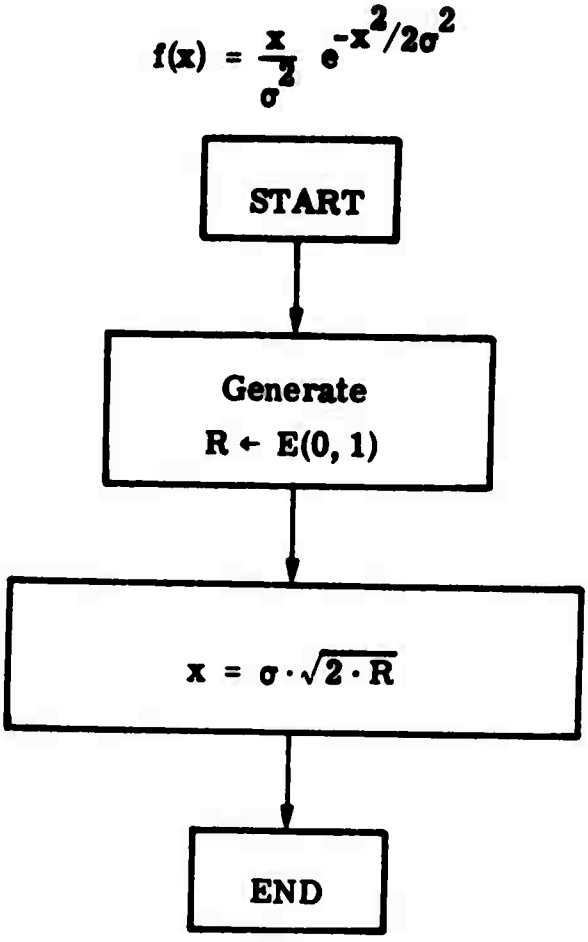


Figure 3-14. Random number generation algorithm for Rayleigh distribution

3.12 GAMMA DISTRIBUTION

The gamma distribution

$$f(x) = \frac{\lambda^{\eta}}{\Gamma(\eta)} x^{\eta-1} e^{-\lambda x} ,$$

describes the time for exactly η events to occur when events occur at a constant rate λ. When η is an integer, there is a simple combination technique for generating gamma variates. However, as the gamma distribution is one of the Pearson family of distributions, there is a need for selecting gamma variates when η is non-integral even though there is no physical model for this. This is a much harder task but can be accomplished by a combination of the usual technique for the integral part of η with a composite rejection technique designed to select from $x^{f} e^{-x}$ where f is the fractional part of η.

Sample routines:

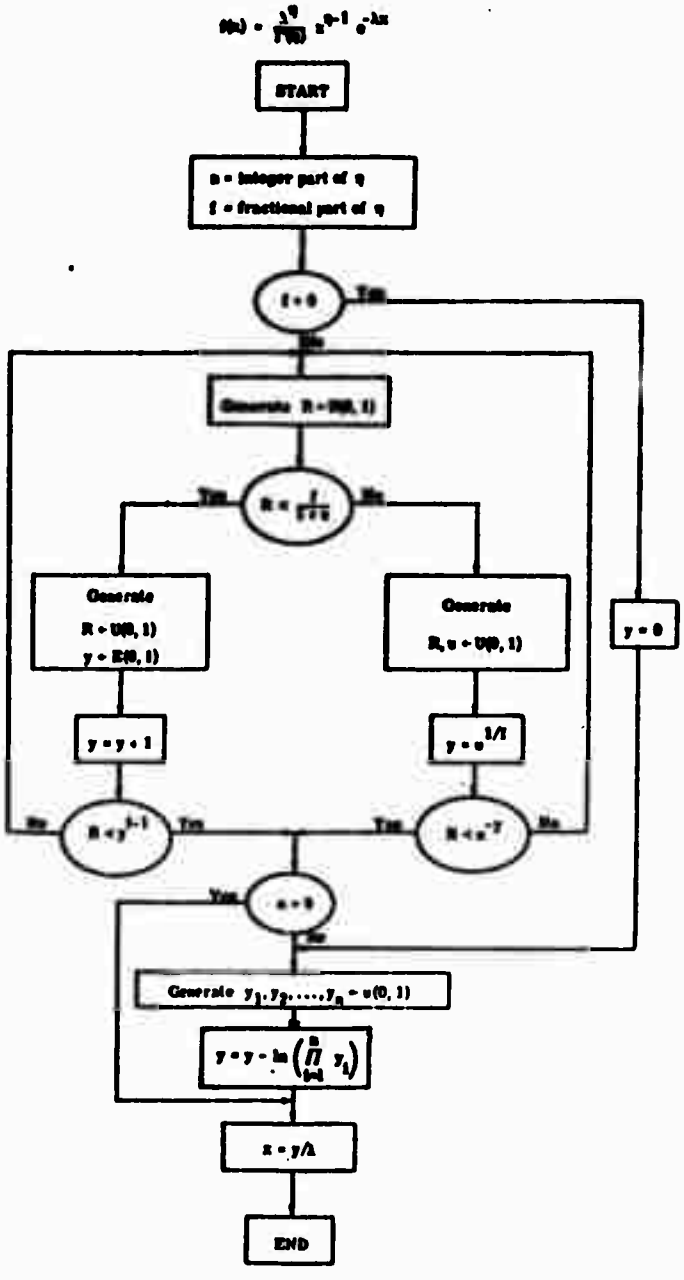
For η integer:

```
      FUNCTION GAMRN (ALAM, NETA)
      Y = 1
      D0 10 I = 1, NETA
10    Y = Y * UNFRN (Y)
      GAMRN = - ALOG(Y)/ALAM
      RETURN
      END
```

For η general:

```
      FUNCTION GAMRN(ALAM, ETA)
      N = ETA
      F = ETA - N
      IF(F.EQ. 0) GO TO 100
10    R = UNFRN(R)
      IF (R.LT. F/(F + 2.71828)) GO TO 20
      Y = UNFRN(Y) ** (1/F)
      IF (UNFRN(R). GT. EXP(-Y)) GO TO 10
      GO TO 50
```

```
100   Y = 0
      GO TO 70
 20   Y = 1. + EXPRN(Y)
      IF(UNFRN(R).GT. Y** (F-1.)) TO TO 10
 50   IF(N.EQ. 0) GO TO 150
 70   Z = 1.0
      DO 80 I = 1, N
 80   Z = Z* UNFRN(Z)
      Y = Y - ALOG(Z)
150   GAMRN = Y/ALAM
      RETURN
      END
```



Note that if q is limited to integral values, this simplifies to:

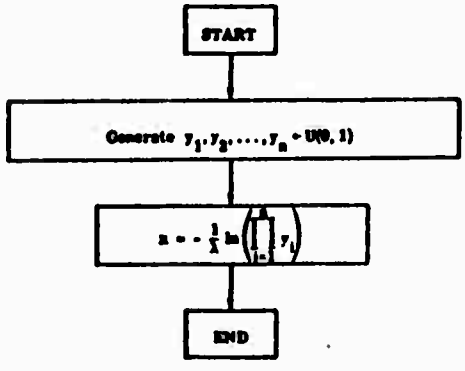


Figure 3-15. Random number generation algorithm for gamma distribution

3.13 BETA DISTRIBUTION

The beta distribution,

$$f(x) = \frac{1}{b-a} \frac{\Gamma(\gamma+\eta)}{\Gamma(\gamma)\Gamma(\eta)} \left(\frac{x-a}{b-a}\right)^{\gamma-1} \left(1-\frac{x-a}{b-a}\right)^{\eta-1} ,$$

with x limited to the interval (a, b), is a basic statistical distribution frequently encountered for bounded variables. The parameters, γ and η, are limited to positive values. Beta variates for most values of the parameters are best obtained as a ratio of two gamma variates. If γ and η are both small integers, a beta variate may also be generated by choosing $\gamma + \eta - 1$ uniform random numbers, arranging them in order of increasing magnitude, and selecting the γ^{th} random number as the beta variate.

Sample routine

```
FUNCTION BETARN (GAM, ETA, A, B)
Y = GAMRN (1., GAM)
Z = GAMRN (1., ETA)
BETARN = (Y/(Y + Z)) * (B - A) + A
RETURN
END
```

$$f(x) = \left(\frac{1}{b-a}\right)\frac{\Gamma(\gamma+\eta)}{\Gamma(\gamma)\Gamma(\eta)}\left(\frac{x-a}{b-a}\right)^{\gamma-1}\left(1-\frac{x-a}{b-a}\right)^{\eta-1} \quad ; \quad a \le x \le b$$

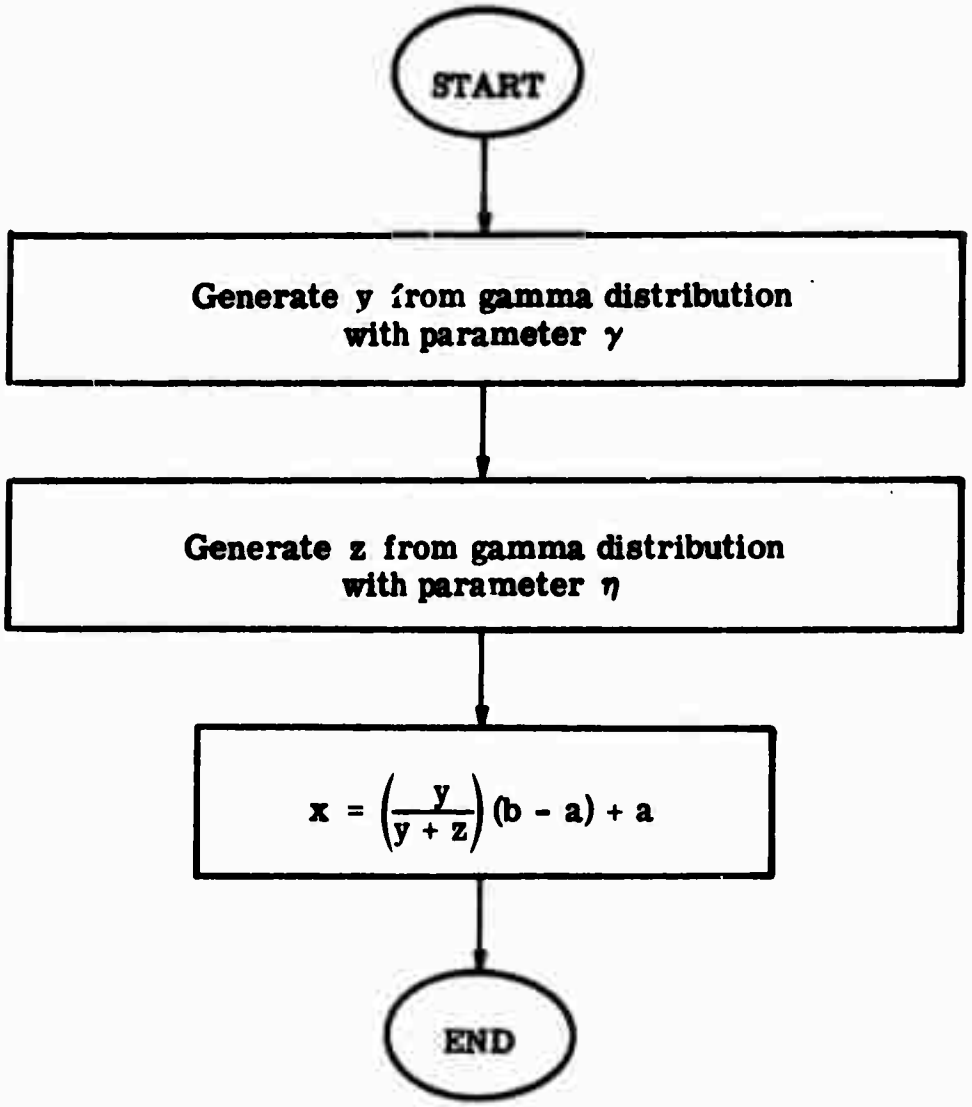


Figure 3-16. Random number generation algorithm for beta distribution

3.14 PARETO DISTRIBUTION

The Pareto distribution, $f(x) = \lambda \epsilon^{\lambda} x^{-\lambda-1}$, has a simple inverse which provides the quickest procedure for random number selection.

Sample routine

```
FUNCTION PRTORN (EPS, ALAM)
PRTORN = EPS * UNFRN(R)**(-1./ALAM)
RETURN
END
```

$$f(x) = \lambda \epsilon^{\lambda} x^{-\lambda-1}$$

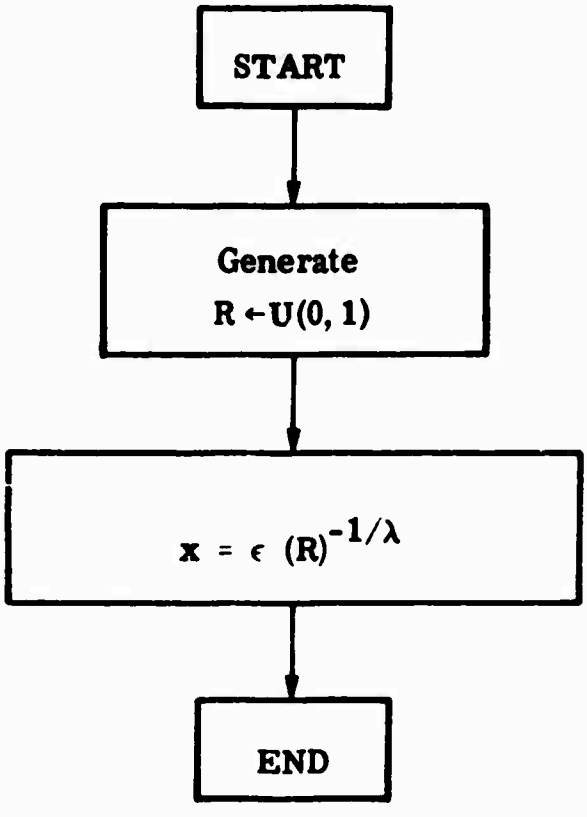


Figure 3-17. Random number generation algorithm for Pareto distribution

3.15 LOG-NORMAL DISTRIBUTION

The log-normal distribution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}(x-\epsilon)} \exp\left\{-\frac{1}{2\sigma^2}[\ln(x-\epsilon)-\mu]^2\right\},$$

describes a random variable whose logarithm is normal. It is a simple matter then to invert this transformation to generate log-normal variates.

Sample routine:

```
FUNCTION ALNMRN (EPS, AMU, SIGMA)
R = ANRMRN(R)
ALNMRN = EPS + EXP (SIGMA*R + AMU)
RETURN
END
```

$$f(x) = \frac{1}{\sigma (x - \epsilon)\sqrt{2\pi}} \exp\left\{-\frac{1}{2\sigma^2}\left[\ln (x - \epsilon) - \mu\right]^2\right\}$$

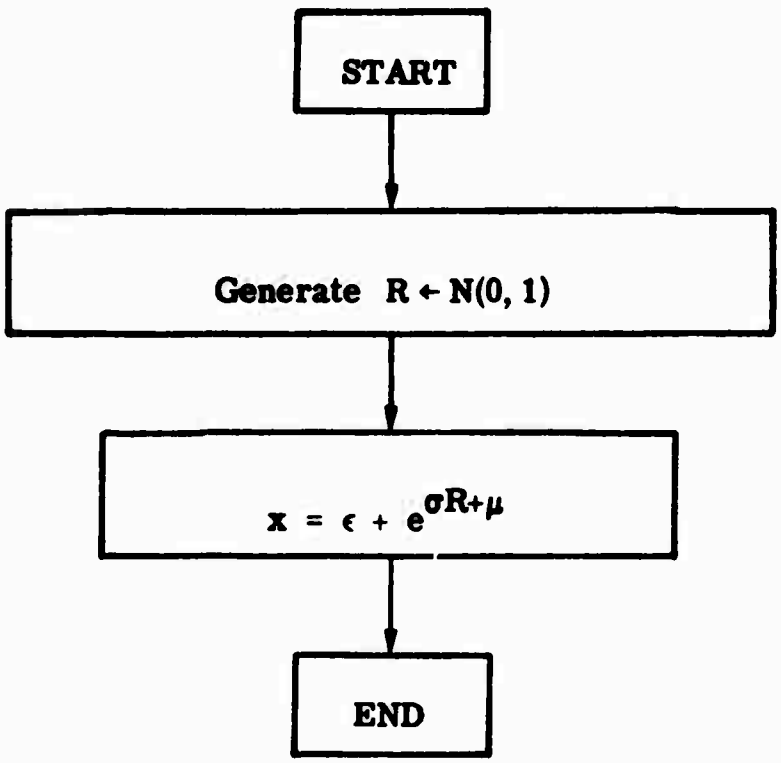


Figure 3-18. Random number generation algorithm for log-normal distribution

3.16 FOLDED-NORMAL DISTRIBUTION

The folded-normal distribution,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\left[e^{-(x-\mu)^2/2\sigma^2} + e^{-(x+\mu)^2/2\sigma^2}\right] ,$$

describes the distribution of the absolute value of a normal variate, which provides the simplest procedure for generating from the distribution.

Sample routine

```
FUNCTION FNRMRN (AMU, SIGMA)
FNRMRN = ABS (AMU + SIGMA * ANRMRN(R))
RETURN
END
```

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\left[e^{-(x-\mu)^2/2\sigma^2} + e^{-(x+\mu)^2/2\sigma^2}\right]$$

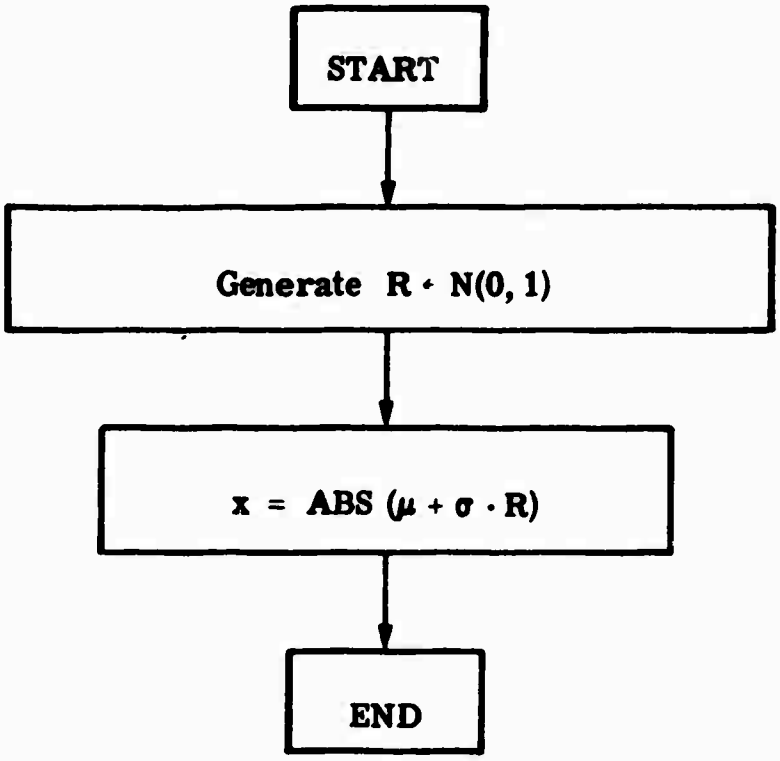


Figure 3-19. Random number generation algorithm for folded-normal distribution

3.17 KODLIN'S DISTRIBUTION

Kodlin suggested as a distribution for survival time data the functional form,

$$f(x) = (\eta + \gamma x)e^{-(\eta x + 1/2\,\gamma x^2)} \quad .$$

This Kodlin form has a moderately simple inverse, and thus it is not difficult to generate random varities.

Sample routine

```
FUNCTION AKODRN (ETA, GAM)
R = EXPRN (R) * 2. * GAM/(ETA **2)
AKODRN = ETA/GAM * (SQRT(1. + R) - 1.)
RETURN
END
```

$$f(x) = (\eta + \gamma x)\, e^{-\left(\eta x + \frac{1}{2}\gamma x^2\right)}$$

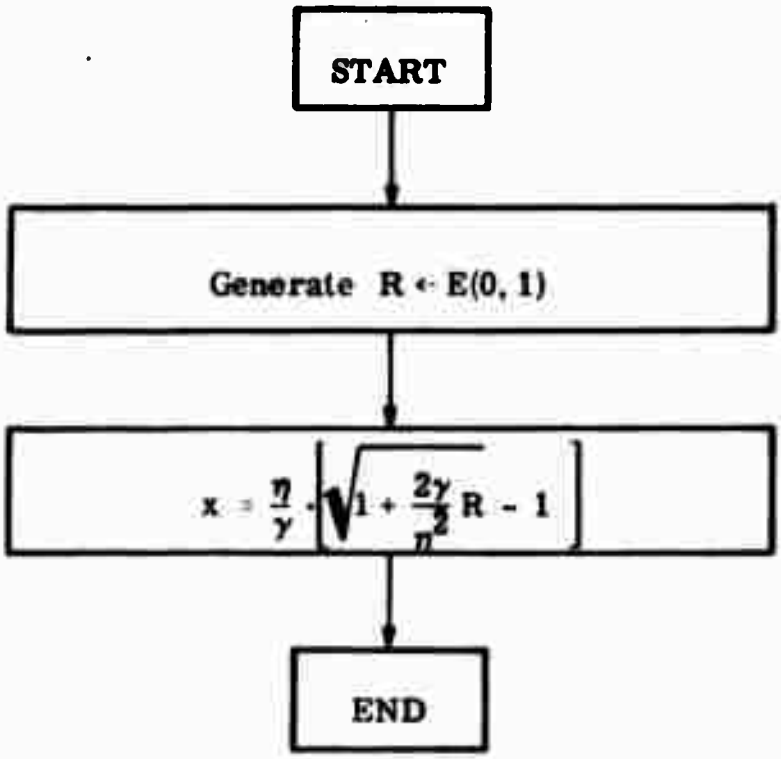


Figure 3-20. Random number generation algorithm for Kodlin's distribution

3.18 EXTREME VALUE DISTRIBUTIONS

There are two extreme value distributions. The first is for the maximum value,

$$f(x) = \frac{1}{\sigma} \exp \left[- \frac{1}{\sigma} (x-\mu) - e^{-(x-\mu)/\sigma} \right] ,$$

and the second is for the minimum value,

$$f(x) = \frac{1}{\sigma} \exp \left[\frac{1}{\sigma} (x-\mu) - e^{(x-\mu)/\sigma} \right] .$$

The inverse function for both is straightforward and provides an efficient selection procedure.

Sample routines

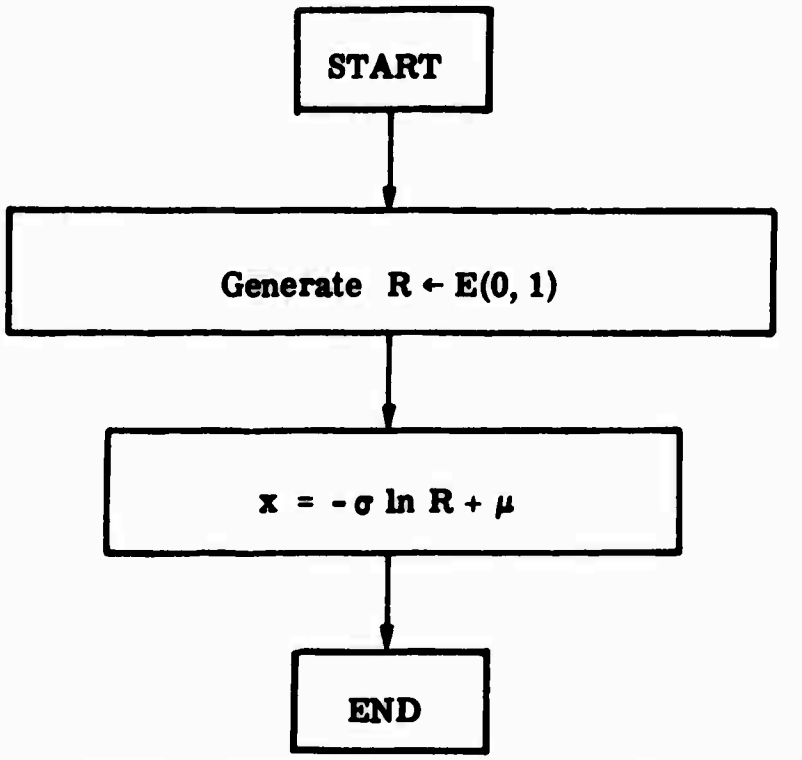
For the maximum value:

```
FUNCTION AMAXRN(AMU, SIG)
R = EXPRN (R)
AMAXRN = AMU - SIG * ALOG (R)
RETURN
END
```

For the minimum value:

```
FUNCTION AMINRN (AMU, SIG)
R = EXPRN (R)
AMINRN = AMU + SIG * ALOG(R)
RETURN
END
```

Maximum value: $f(x) = \frac{1}{\sigma} \exp\left[-\frac{1}{\sigma}(x-\mu) - e^{-(x-\mu)/\sigma}\right]$



Minimum value: $f(x) = \frac{1}{\sigma} \exp\left[\frac{1}{\sigma}(x-\mu) - e^{(x-\mu)/\sigma}\right]$

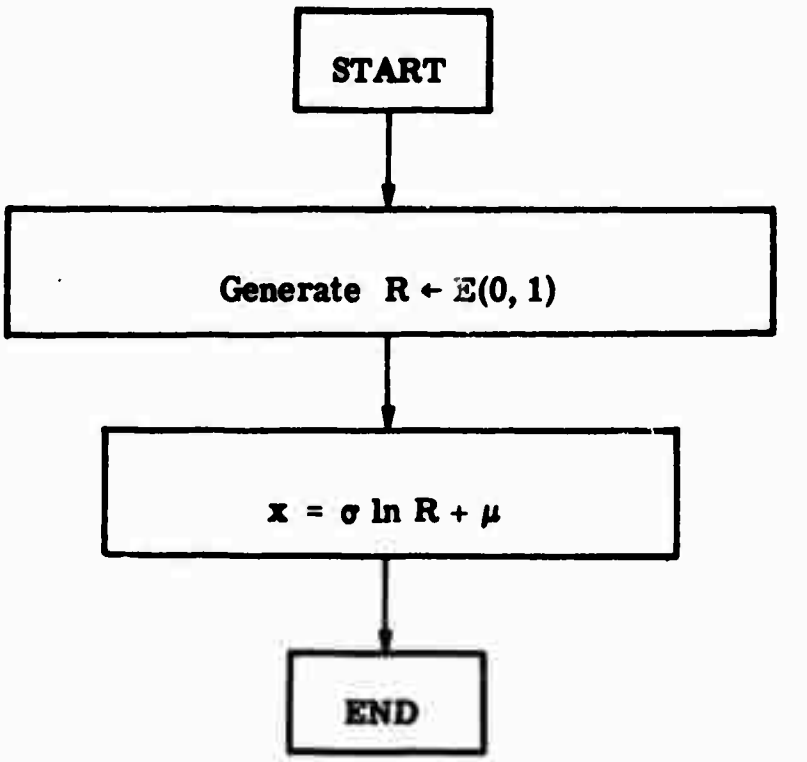


Figure 3-21. Random number generation algorithm for extreme value distributions

3.19 WEIBULL DISTRIBUTION

The Weibull distribution, $f(x) = \eta/\lambda\,(x-\epsilon)^{\eta-1} \exp[-(x-\epsilon)^{\eta}/\lambda\,]$, is a three-parameter $(\epsilon, \lambda, \eta)$ family of empirical distributions having wide usefulness. The random variable x is bounded below by ϵ. The inverse cumulative function is straightforward and provides the best general method for generating Weibull random numbers.

Sample routine:

```
FUNCTION WIBLRN (EPS, ALAM, ETA)
WIBLRN = EPS + (ALAM * EXPRN (ALAM)) ** (1. /ETA)
RETURN
END
```

$$f(x) = \frac{\eta}{\lambda}(x - \epsilon)^{\eta-1} \; e^{-\frac{(x-\epsilon)^{\eta}}{\lambda}}$$

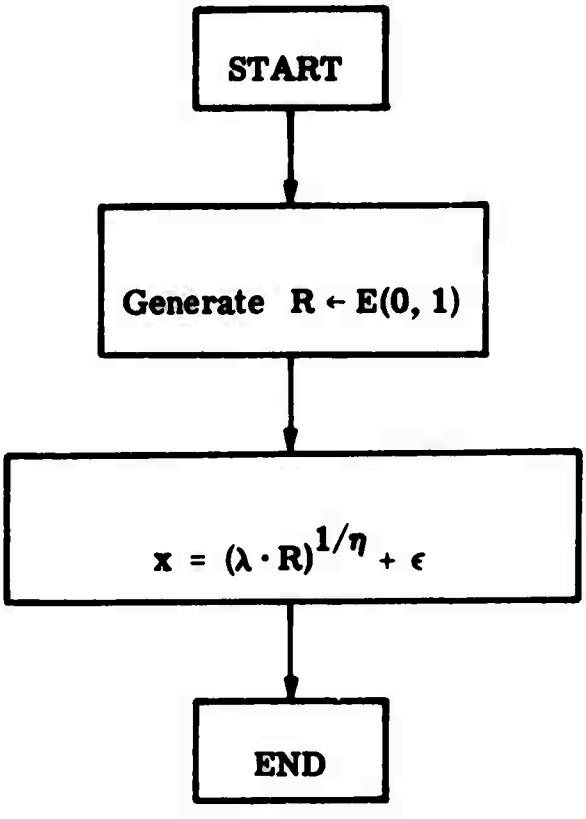


Figure 3-22. Random number generation algorithm for Weibull distribution

3.20 JOHNSON DISTRIBUTIONS

3.20.1 Johnson S_L Distribution

$$f(x) = \frac{\eta}{\sqrt{2\pi}(x-\epsilon)} \exp\left\{-\frac{\eta^2}{2}\left[\frac{\gamma}{\eta} + \ell n(x-\epsilon)\right]^2\right\} ,$$

is easily generated by transforming a normal variate. (The reverse of the transformation used in deriving this Johnson distribution.) The S_L distribution is also known as the log-normal (Section 3.15).

Sample routine:

```
FUNCTION SLRN (EPS, GAM, ETA)
R = ANRMRN (R)
SLRN = EPS + EXP ((R-GAM)/ETA)
RETURN
END
```

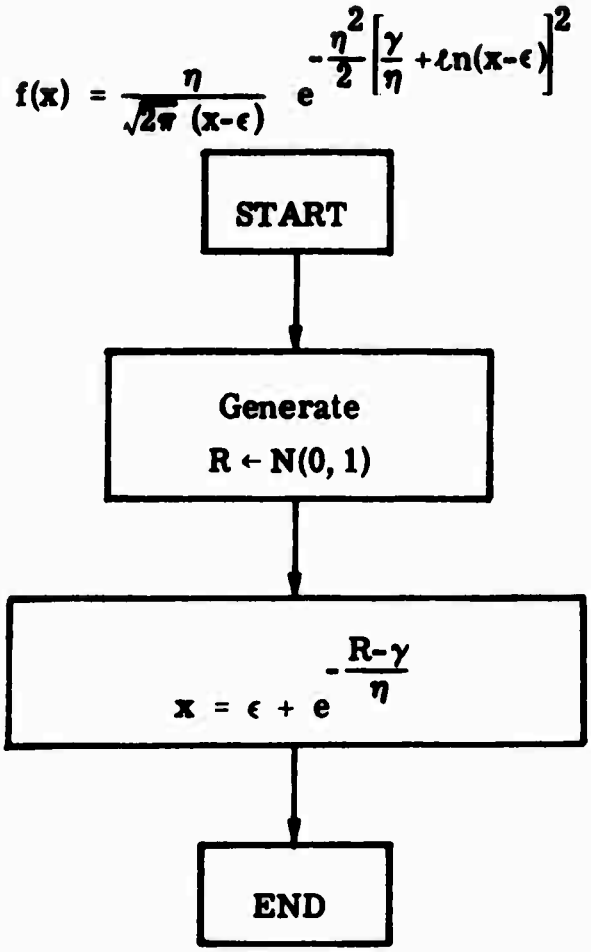


Figure 3-23. Random number generation algorithm for Johnson S_L distribution

3.20.2 Johnson S_B Distribution

The Johnson S_B distribution,

$$f(x) = \frac{\eta}{\sqrt{2\pi}} \frac{\lambda}{(x-\epsilon)(\lambda-x+\epsilon)} \exp\left\{-\frac{1}{2}\left[\gamma+\eta \ln\left(\frac{x-\epsilon}{\lambda-x+\epsilon}\right)\right]^2\right\},$$

is easily generated by a transformation on a normal variate.

Sample routine:

```
FUNCTION SBRN (EPS, ALAM, GAM, ETA)
R = ANRMRN (R)
EX = EXP ((R-GAM)/ETA)
SBRN = EPS + ALAM * EX/(1. + EX)
RETURN
END
```

$$f(x) = \frac{\eta}{\sqrt{2\pi}} \frac{\lambda}{(x-\epsilon)(\lambda-x+\epsilon)} e^{-\frac{1}{2}\left|\gamma+\eta \ln\left(\frac{x-\epsilon}{\lambda-x+\epsilon}\right)\right|^2}$$

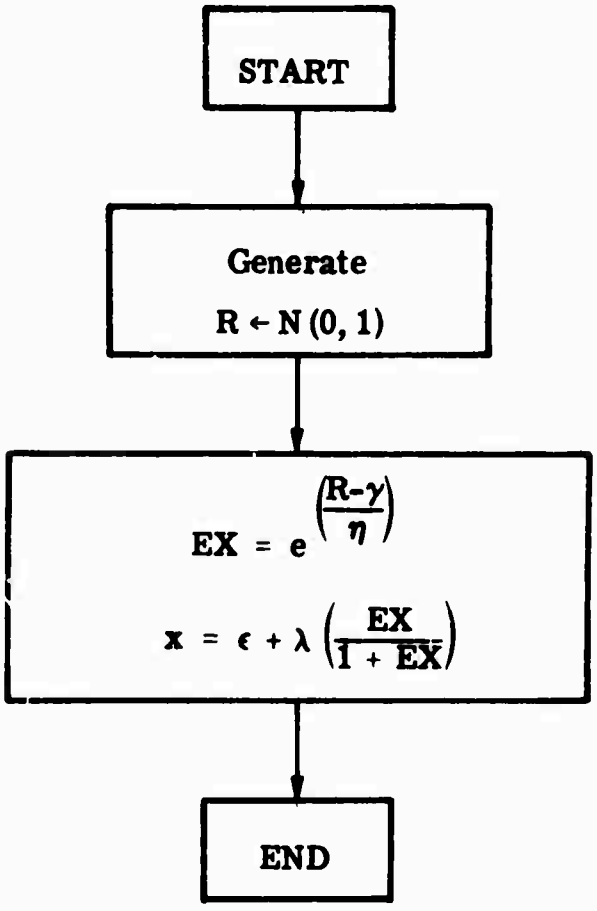


Figure 3-24. Random number generation algorithm for Johnson S_B distribution

3.20.3 Johnson S_U Distribution

Like the other Johnson family distributions, the S_U distribution,

$$f(x) = \frac{\eta}{\sqrt{2\pi}} \frac{1}{\sqrt{(x+\epsilon)^2+\lambda^2}} \exp\left[-\frac{1}{2}\left(\gamma+\eta \ln\left\{\left(\frac{x-\epsilon}{\lambda}\right)+\left[\left(\frac{x-\epsilon}{\lambda}\right)^2+1\right]^{1/2}\right\}\right)^2\right] ,$$

is easily selected by reversing the transform which generated the distribution from a normal distribution.

Sample program:

```
FUNCTION SURN (EPS, ALAM, GAM, ETA)
R = ANRMRN(R)
SURN = EPS + ALAM * SINH ((R - GAM)/ETA)
RETURN
END
```

$$S_u: \quad f(x) = \frac{\eta}{\sqrt{2\pi}} \frac{1}{\sqrt{(x+\epsilon)^2 + \lambda^2}} \; e^{-\frac{1}{2}\left(\gamma + \eta \ln\left\{\left(\frac{x-\epsilon}{\lambda}\right) + \left[\left(\frac{x-\epsilon}{\lambda}\right)^2 + 1\right]^{1/2}\right\}\right)^2}$$

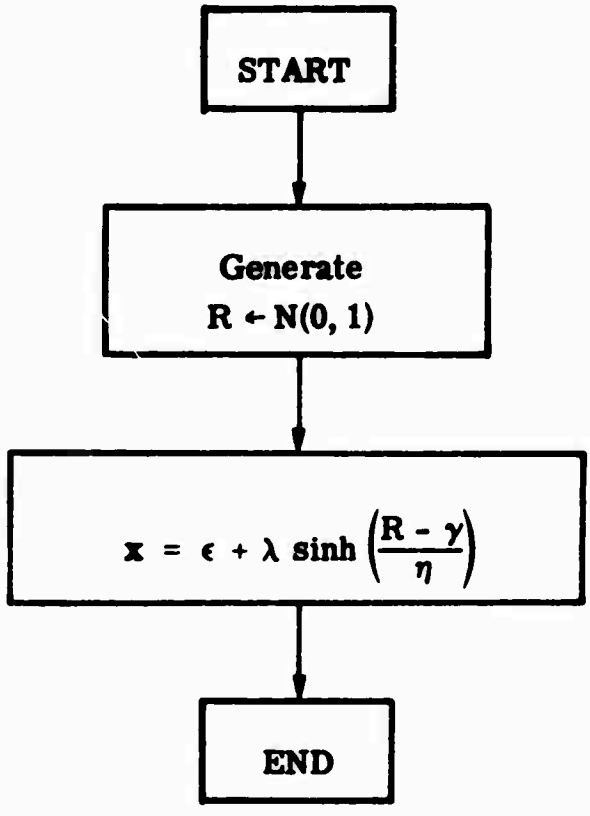


Figure 3-25. Random number generation algorithm for Johnson S_U distribution

3.21 PEARSON DISTRIBUTIONS

3.21.1 Pearson Type I Distribution

The Type I distribution of the Pearson system of frequency functions is given by

$$f(x) = C(1 + x/a_1)^{m_1} (1 - x/a_2)^{m_2} ,$$

where C is a normalization constant. The limits on the distribution are $-a_1 < x < a_2$ and there are further constraints that $m_1 > -1$ and $m_2 > -1$. By the linear transformation $Z = \frac{x + a_1}{a_2 + a_1}$, the Type I distribution can be transformed into a beta distribution which may be derived from gamma variates as given in Section 3.13.

Sample routine:

```
FUNCTION TYP1RN(EM1, EM2, A1, A2)
U = GAMRN (1., EM1+1.)
V = GAMRN (1., EM2+1.)
TYP1RN = (A1 + A2)*U/(U+V) - A1
RETURN
END
```

$$f(x) = C(1 + x/a_1)^{m}1\ (1 - x/a_2)^{m}2$$

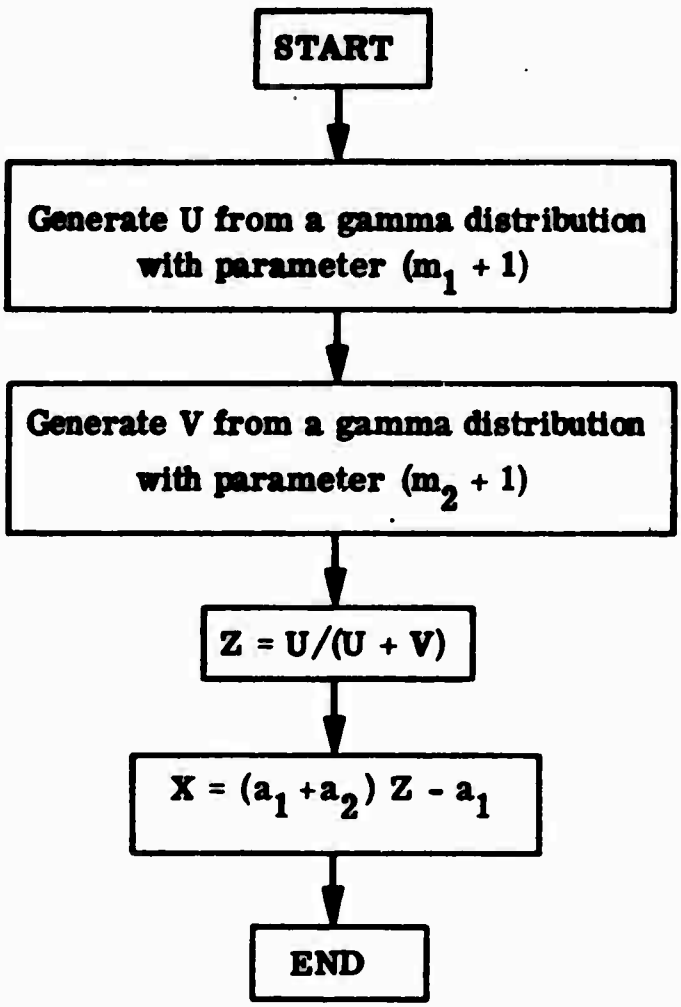


Figure 3-26. Random number generation algorithm for the Pearson Type 1 distribution

3.21.2 Pearson Type II Distribution

The second distribution in the Pearson family is given by

$$f(x) = C\,(1 - \frac{x^2}{a^2})^m \quad ,$$

where C is a normalization factor. The limits on the distribution are $-a < x < a$ and $m > -1$. This is a special case of Type I where $m_1 = m_2$ and $a_1 = a_2$. As such it may also be derived from gamma variates.

Sample routine:

```
FUNCTION TYPE2RN(EM,A)
U = GAMRN (1., EM+1)
V = GAMRN (1., EM+1)
TYP2RN = A*(U-V)/(U+V)
RETURN
END
```

$$f(x) = C\left(1 - \frac{x^2}{a^2}\right)^m$$

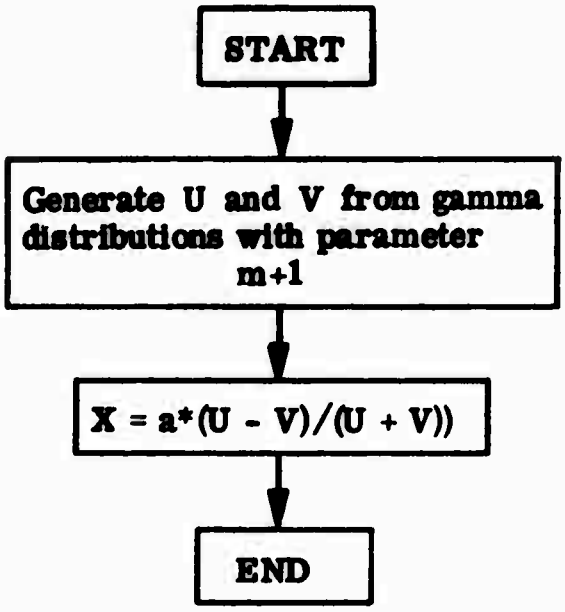


Figure 3-27. Random number generation algorithm for the Pearson Type II distribution

3.21.3 Pearson Type III Distribution

The Pearson Type III distribution is given by $f(x) = C(1 + x/a)^{\gamma a} e^{-\gamma x}$, where C is a normalization constant. The distribution is limited to $-a < x < a$ (or to $a < x < -a$ if a is negative) and is further constrained by $\gamma a > -1$. A few simple transformations, $x = a(y-1)$ and $\lambda = a\gamma$, will turn this distribution into a special form of the gamma distribution $f(y) = C' y^{\lambda} e^{-\lambda y}$.

Sample routine:

```
FUNCTION TYP3RN (GAM, A)
P = GAM*A
Y - GAMRN(P, P+1.)
TYP3RN = A*(Y-1.)
RETURN
END
```

$$f(x) = C\ (1 + x/a)^{\gamma a}\ e^{-\gamma x}$$

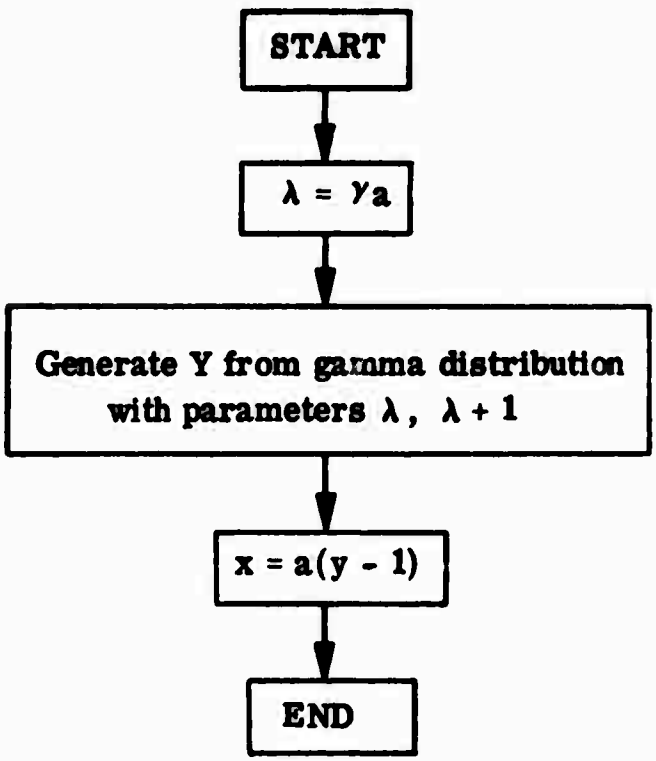


Figure 3-28. Random number generation algorithm for the Pearson Type III distribution

3.21.4 Pearson Type IV Distribution

The Type IV distribution of the Pearson system is given by

$$f(x) = C\,(1 + x^2/a^2)^{-m} e^{-\gamma \tan^{-1}(x/a)} \quad ,$$

where C is a normalization constant. By a trigonometric transformation, $x = a \tan^{-1}(\varphi - \pi/2)$, the function can be transformed into $f(\varphi) = C'(\sin\varphi)^r e^{-\gamma\varphi}$, where $\gamma = 2m - 2$. In this form there is one limit on the parameters, namely $r > 3$, while φ ranges from 0 to π. Picking from this function can be accomplished by a selection from $e^{-\gamma\varphi}$, truncated at $\varphi = \pi$, followed by a rejection conditioned on $(\sin\varphi)^r$.

Sample routine:

```
      FUNCTION TYP4RN(EM,GAMMA,A)
      DATA PI/3.1415962/HAFPI/1.5707981/
      R = 2*EM-2
10    PHI = EXPRN(R)
      PHI = AMOD(PHI/GAMMA,PI)
      IF (UNFRN(R).GT.(SIN(PHI)**R)) GO TO 10
      TYP4RN = A*TAN(PHI-HAFPI)
      RETURN
      END
```

$$f(x) = C\,(1 + x^2/a^2)^{-m_e - \gamma \tan^{-1}(x/a)}$$

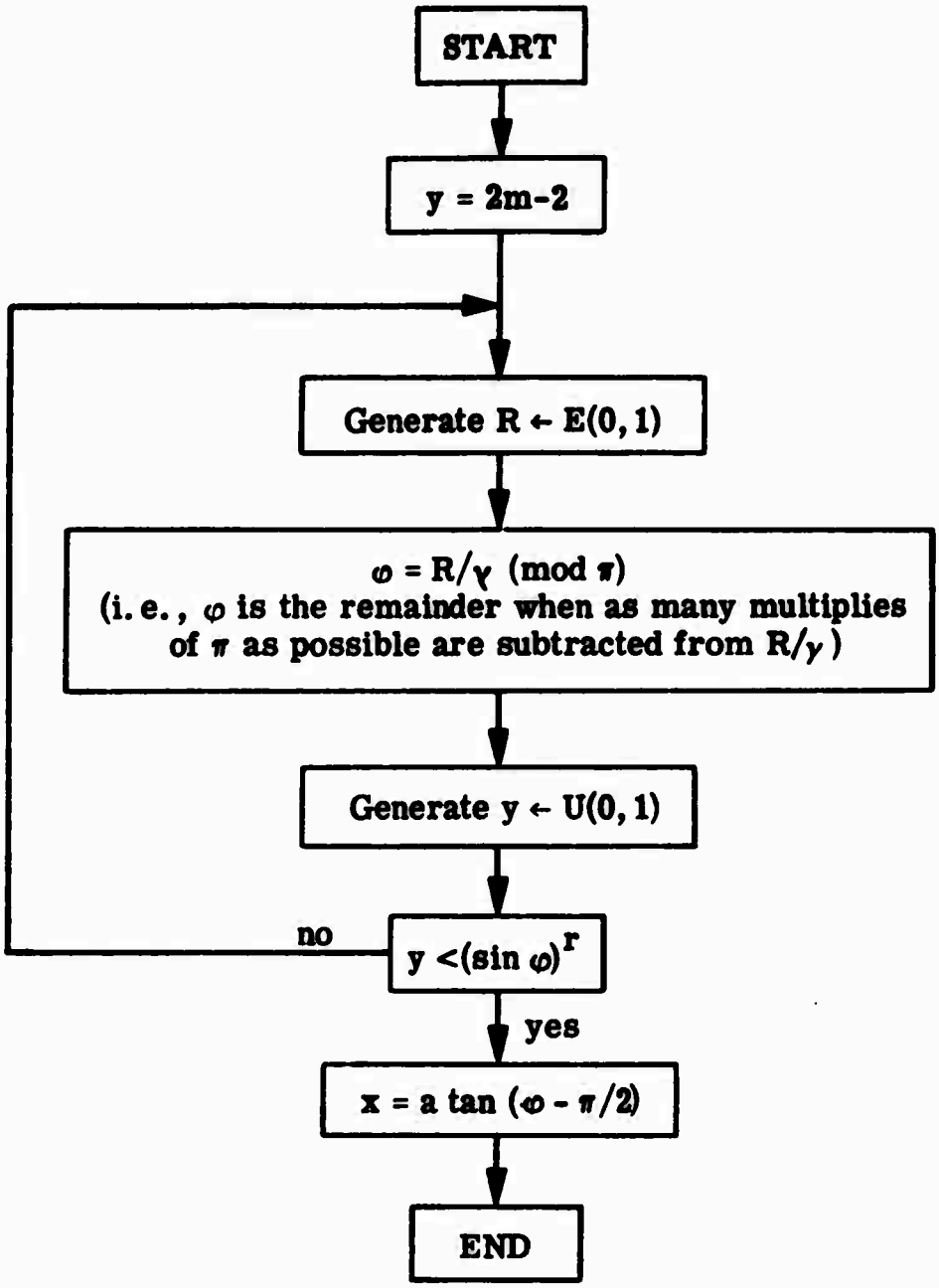


Figure 3-29. Random number generation algorithm for the Pearson Type IV distribution

3.21.5 Pearson Type V Distribution

The fifth type of distribution in the Pearson system of frequency functions is given by $f(x) = C\, x^{-p} e^{-\gamma/x}$, where C is a normalization constant. The range of the argument is $0 < x < \infty$. The parameter γ must be positive (for $\gamma < 0$, $-\infty < x < 0$) , and p must be greater than 1 . The Type V random variable x is the inverse of a gamma variate; this provides the simplest means of picking from the Type V distribution.

Sample routine:

```
FUNCTION TYP5RN (P,GAMMA)
TYP5RN = 1./GAMRN(GAMMA,P-1.)
RETURN
END
```

$$f(x) = C\, x^{-p}\, e^{-\gamma/x}$$

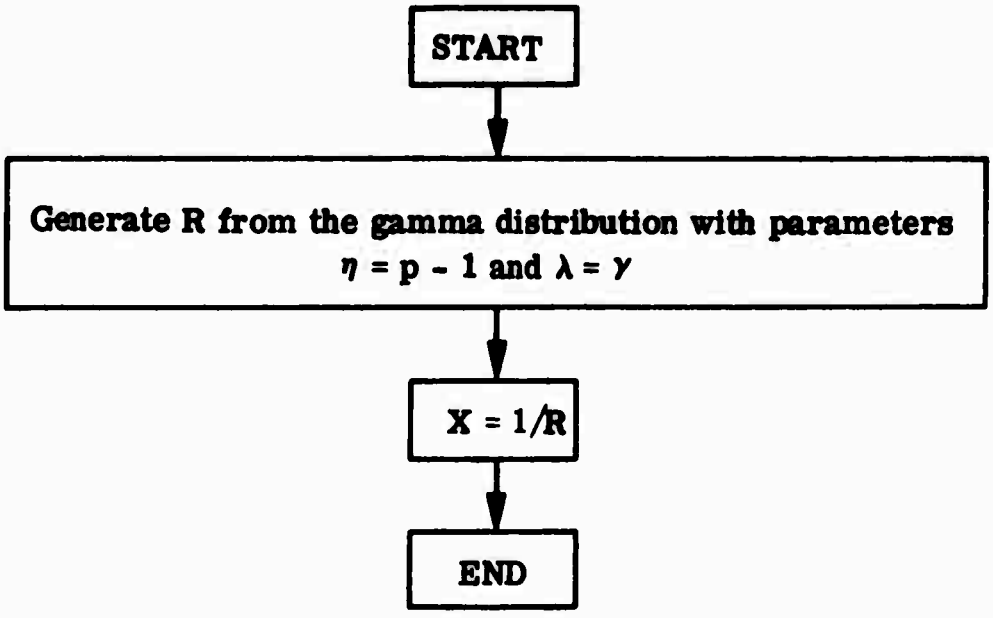


Figure 3-30. Random number generation algorithm for the Pearson Type V distribution

3.21.6 Pearson Type VI Distribution

Type VI of the Pearson family of distributions is given by $f(x) = C(x-a)^{q_2} x^{-q_1}$, where C is a normalization factor and q_1 and q_2 are parameters limited by $q_1 > q_2 + 1 > 0$. For $a > 0$ the range of the distribution is $a < x < \infty$ while for negative a it is $-\infty < x < a$. By the simple transformation $x = a/y$ the distribution is converted into a form of the beta distribution

$$f(y) = C' y^{(q_1 - q_2 - 2)} (1-y)^{q_2} \qquad 0 < y < 1$$

which can be obtained from two gamma variates as described in 3.13.

Sample routine:

```
FUNCTION TYP6RN(A,Q1,Q2)
U = GAMRN(1.,Q1-Q2-1.)
V = GAMRN(1.,Q2+1.)
TYP6RN = A*(U+V)/U
RETURN
END
```

$$f(x) = C(x-a)^{q_2} x^{-q_1}$$

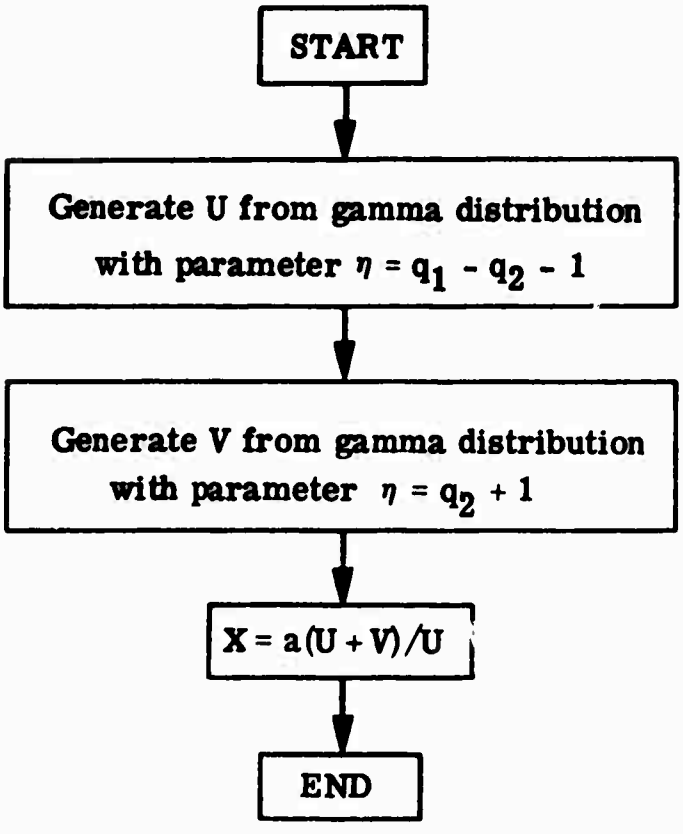


Figure 3-31. Random number generation algorithm for the Pearson Type VI distribution

3.21.7 Pearson Type VII Distribution

Type VII of the Pearson family of distributions is given by

$$f(x) = C\,(1 + x^2/a^2)^{-m},$$

where C is a normalization factor. The range of x is $-\infty$ to ∞ where m must be greater than 2.5. By setting $z = \frac{a^2}{a^2 + x^2}$ the distribution is transformed into

$$g(z) = C'\,(1 - z)^{-1/2}\, z^{m-3/2}$$

which is a special case of a Beta distribution with $\gamma = m-1/2$ and $\eta = 1/2$. The beta variate z can be obtained as a ratio of two gamma variates, $z = u/(u+v)$. As $x = a(1/z - 1)^{1/2}$, we have $x = a\,(v/u)^{1/2}$. Now v is a gamma variate with parameter $\eta = 1/2$. This special case of a gamma variate can be obtained from $v = y^2/2$, where y is a normalized normal variate. This gives $x = ay\,(1/2u)^{1/2}$. Selection from the Pearson Type VII is achieved by combining the above transformations with the selection routines for the gamma and normal variates.

Sample Routine

```
FUNCTION TYP7RN(A, EM)
Y = ANRMRN(Y)
U = GAMRN (.5, EM -.5)
TYP7RN = A*Y/SQRT(U)
RETURN
END
```

$$f(x) = C\,(1 + x^2/a^2)^{-m}$$

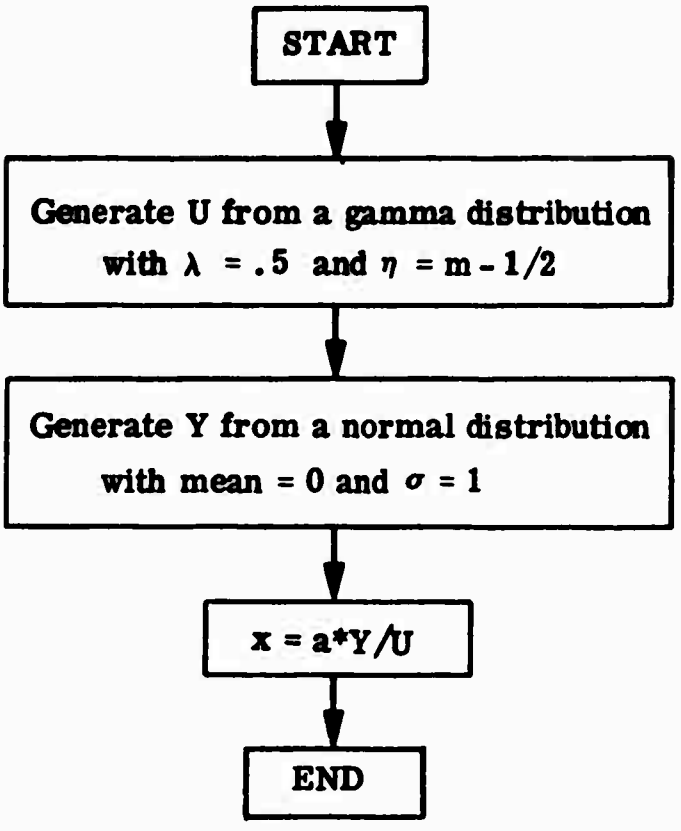


Figure 3-32. Random number generation algorithm for the Pearson Type VII distribution

3.21.8 Type VIII Pearson Distribution

The eighth distribution in the Pearson family is given by

$$f(x) = C\ (1 + x/a)^{-m}$$

where C is a normalization constant. The range of x is $-a < x < 0$ (or $0 < x < -a$ for a negative) while the range of m is $0 \leq m \leq 1$. If we set $y = (1 + x/a)$, the distribution becomes

$$f(y) = C'\ y^{-m} \quad \text{where} \quad 0 < y < 1.$$

This form of the distribution has a simple inverse.

Sample Routine

```
FUNCTION TYP8RN(A, EM)
R = UNFRN(R)
TYP8RN = A*(R**(1./(1.EM)) - 1.)
RETURN
END
```

$$f(x) = C\,(1 + x/a)^{-m}$$

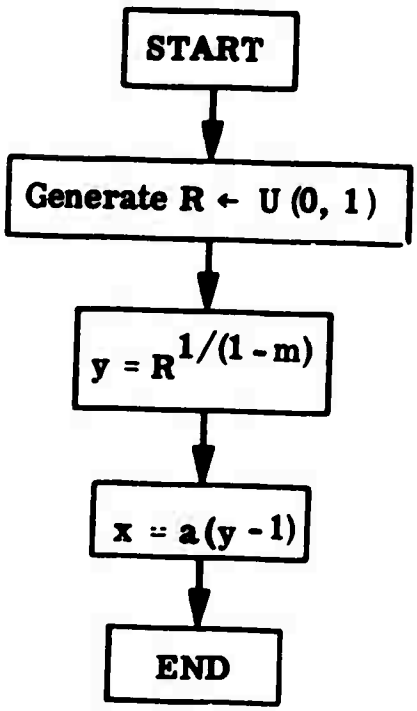


Figure 3-33. Random number generation algorithm for the Pearson Type VIII distribution

3.21.9 Pearson Type IX Distribution

The Pearson Type IX distribution is given by

$$f(x) = C\,(1 + x/a)^{m},$$

where C is the normalization factor. The range of x is -a to 0 while m must be greater than zero. This function has a simple inverse.

Sample Routine

```
FUNCTION TYPE9RN(A, EM)
R = UNFRN(R)
TYP9RN = A*(R** (1./(EM + 1.))-1.)
RETURN
END
```

$$f(x) = C\ (1+x/a)^m$$

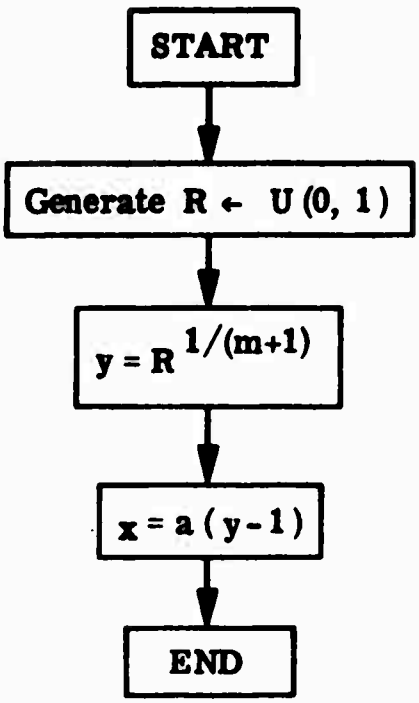


Figure 3-34. Random number generation algorithm for the Pearson Type IX distribution

3.21.10 Pearson Type X Distribution

The Pearson Type X distribution is a form of the exponential distribution given by

$$f(x) = 1/\sigma \; e^{-x/\sigma} \quad ; \quad x \geq 0$$

This is easily obtained from the standard exponential distribution routines.

Sample Routine

```
FUNCTION TP10RN(SIGMA)
TP10RN = SIGMA*EXPRN(SIGMA)
RETURN
END
```

$$f(x) = 1/\sigma \; e^{-x/\sigma}$$

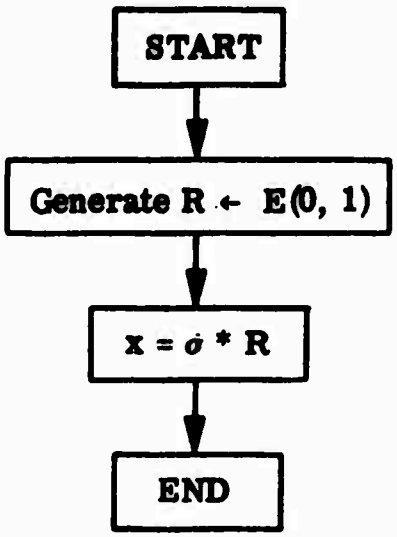


Figure 3-35. Random number generation algorithm for the Pearson Type X distribution

3.21.11 Pearson Type XI Distribution

The eleventh in the series of Pearson distribution is given by

$$f(x) = C(b/x)^m$$

where C is a normalization factor. The range of x is limited to $b < x < \infty$. The parameter m is greater than 1. This distribution has a simple inverse.

Sample Routine

```
FUNCTION TP11RN(B, EM)
R = UNFRN(R)
Y = R**(1./(EM-1.))
TP11RN = B/Y
RETURN
END
```

$$f(x) = C(b/x)^m$$

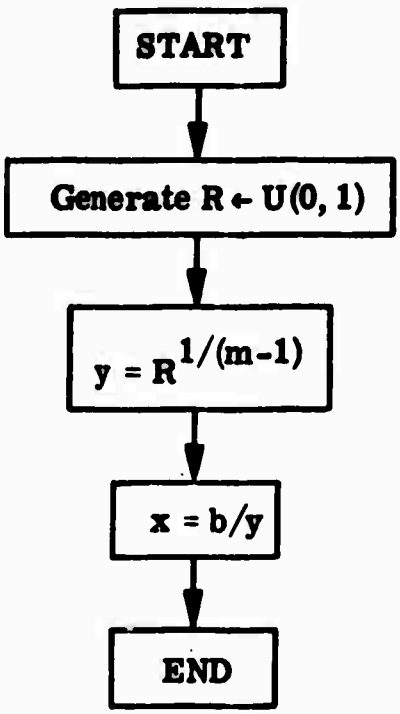


Figure 3-36. Random number generation algorithm for the Pearson Type XI distribution

3.21.12 Pearson Type XII Distribution

Type XII of the Pearson system of distributions is given by

$$f(x) = C\left[\frac{\sigma(\sqrt{3+\beta_1} + \sqrt{\beta_1}) + x}{\sigma(\sqrt{3+\beta_1} - \sqrt{\beta_1}) - x}\right]^{(\sqrt{\beta_1/(3+\beta_1)})}$$

where C is a normalization factor, σ is the standard deviation, and $\beta_1 = \mu_3^2/\mu_2^3$ (skewness). The range of x is

$$-\sigma(\sqrt{3+\beta_1} + \sqrt{\beta_1}) < x < \sigma(\sqrt{3+\beta_1} - \sqrt{\beta_1}) \quad .$$

By setting

$$m = \sqrt{\beta_1/(3+\beta_1)} \ ,$$

$$a = \sigma(\sqrt{3+\beta_1} + \sqrt{\beta_1}) \ , \text{ and}$$

$$b = \sigma(\sqrt{3+\beta_1} - \sqrt{\beta_1}) \ ,$$

the distribution becomes

$$f(x) = C\left(\frac{a+x}{b-x}\right)^m \quad .$$

By setting

$$y = \frac{x+a}{b+a} \ ,$$

the distribution transforms to $f(y) = C' y^m (1-y)^{-m}$ which is a special case of the Beta distribution.

Sample Routine

```
FUNCTION TY12RN(SIGMA, BETA1)
R = SQRT(BETA1)
S = SQRT(BETA1+3)
EM = R/S
A = SIGMA*(R+S)
B = SIGMA*(S-R)
Y = BETARN(EM+1, 1-EM)
TP12RN = (B+A)*Y-A
RETURN
END
```

$$f(x) = C\left[\frac{\sigma(\sqrt{3+\beta_1}+\sqrt{\beta_1})+x}{\sigma(\sqrt{3+\beta_1}-\sqrt{\beta_1})-x}\right]^{\sqrt{\beta_1/(3+\beta_1)}}$$

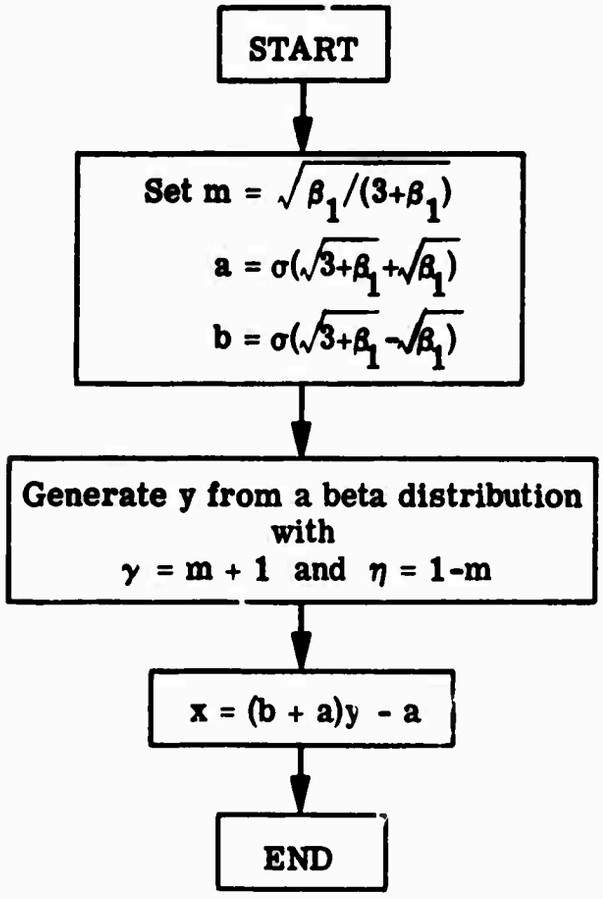


Figure 3-37. Random number generation algorithm for the Pearson Type XII distribution

3.22 HISTOGRAM DISTRIBUTIONS

Frequently, empirical data regarding a probability distribution is obtained in a histogram form. That is, intervals (x_o, x_1), (x_1, x_2), ... (x_{n-1}, x_n) and probabilities $p_1, p_2, \ldots, p_n$ are given such that p_i is the probability that the variable x is found in the interval from x_{i-1} to x_i. (It is presumed that the histogram is normalized, i.e. $\sum_{i=1}^{n} p_i = 1.$) Within each interval it is assumed that the probability is constant.

Selecting a random number from such a histogram distribution is simple. It is necessary first to select the interval in which the random number falls, and then to choose where in that interval the random number lies. This is basically an inverse distribution technique. Selection of the interval i is accomplished by generating a uniform random number and subtracting off successive values of p_i. The value of i when this result first goes negative is the desired interval index. Generation of a second uniform random number and scaling it to fit in the interval from x_{i-1} to x_i completes the task.

A more efficient (much more efficient if the size of the data table is large) generator can be produced if it is possible to cast the histogram data in a form such that $p_1 = p_2 = \ldots = p_n = 1/n$ by choosing values of x_i appropriately. Such a representation is known as equal probability bins. This greatly simplifies selection of the interval i as all n intervals have the same probability. Successive subtraction of values of p_i is no longer needed and can be replaced by a direct calculation of i from a uniform random number.

In the sample Fortran routines below, the array X(I) is presumed to contain: $X(1) = x_0$, $X(2) = x_1$, ..., $X(N + 1) = x_n$. In the first routine use is made of the fact that, at the conclusion of selection of i, R will be uniformly distributed between 0 and $- p_i$.

Sample Routines

For general histogram selection

```
      FUNCTION HISTRN (N, X, P)
      DIMENSION X(N), P(N)
      R = UNFRN (R)
      DO 10 I = 1, N
      R = R - P(I)
      IF (P . LT . 0) GO to 20
10    CONTINJE
20    HSTRN = X(I) - R * (X (I + 1) - X (I))/P(I)
      RETURN
      END
```

For selection with an equal probability bin histogram

```
      FUNCTION HSTRN (N, X)
      DIMENSION X(N)
      R = N * UNFRN (R) + 1
      I = R
      R = R - I
      HSTRN = X(I) + R * (X(I + 1) - X(I))
      RETURN
      END
```

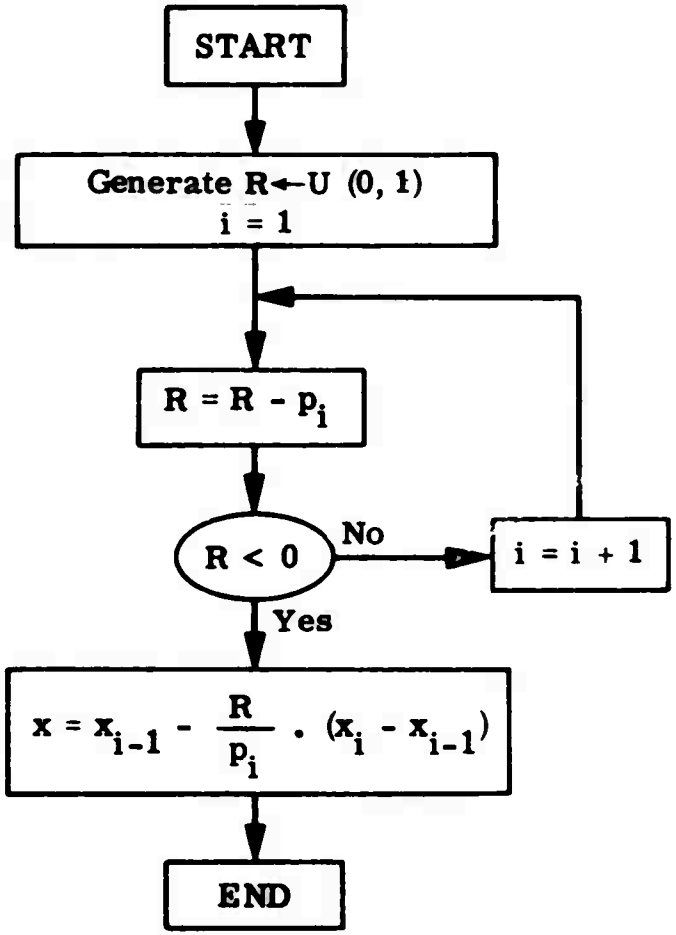


Figure 3-38. Random number generation algorithm for a histogram distribution

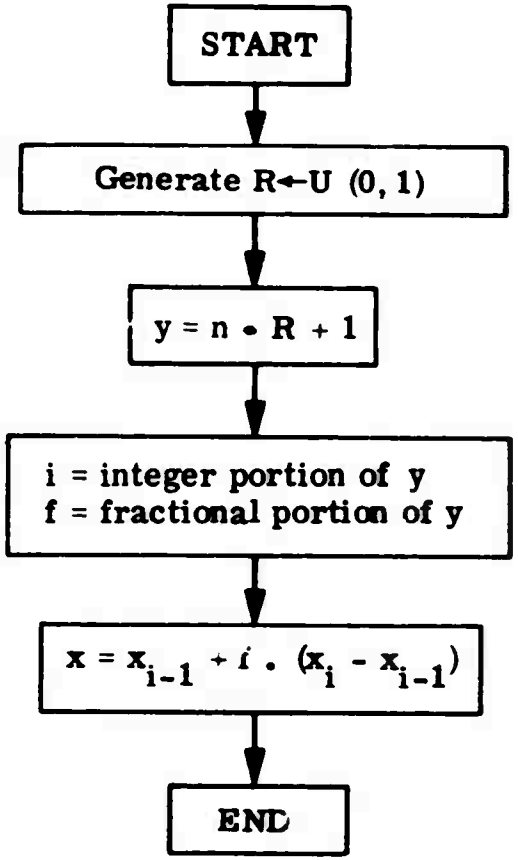


Figure 3-39. Random number generation algorithm for an equal probability bin histogram distribution

APPENDIX A

GENERAL TECHNIQUES FOR GENERATING RANDOM NUMBERS FROM DESIRED DISTRIBUTIONS

When given a particular distribution, $f(x)$, and the task of selecting random numbers distributed according to that function, the investigator has a large number of possible alternatives at his disposal. The primary task is to derive a method which will accomplish the desired selection. A secondary task is to choose the method which is least time-consuming computationally.

Unfortunately, it is not possible to give a straightforward methodology for deriving random number generation techniques which can be applied in all or even in most cases. The situation closely parallels that of finding an integral of an arbitrary function. When one encounters the need to integrate an unfamiliar function, the first step, of course, is to try to look it up in a table of integrals. That failing,one must try to simplify, transform variables, integrate by parts, use trigonometric substitutions, or employ other similar tricks to reduce the integral to a familiar form. There is no guarantee of success, and much depends on the ingenuity and experience of the researcher. When all else fails you can "grind out" a numerical solution.

Faced with the task of generating random numbers from an unfamiliar distribution, a similar procedure is needed. The first step is to try to look it up somewhere — such as in Section 3 of this report. If not found there, there are a number of techniques — inverse, rejection, transformations, combinations, etc. available. These are described in this Appendix. There is no guarantee of success in using them, and the experience and ingenuity

of the analyst is very important. As a final resort, there are numerical methods which can be applied.

The following description of general techniques, while not universally applicable should give the reader some notion of how to proceed in deriving random number generation algorithms.

A.1 THE INVERSE METHOD[6]

The first technique which one should consider is the inverse. To apply the inverse method, the distribution function is integrated to give the cumulative distribution, $F(x) = \int_{-\infty}^{x} f(x')dx'$. This is the probability of selecting a number less than or equal to x. This is equated to the probability of selecting a random number, R, from the uniform distribution. Thus, $F(x) = \int_{-\infty}^{x} f(x')\,dx' = R$. The question then is whether or not this equation has a simple closed-form solution, $x = F^{-1}(R)$. If the inverse function exists, then it is a solution to our task, for, if R is distributed uniformly, then $x = F^{-1}(R)$ is distributed according to f(x). If $F^{-1}(R)$ not only exists,but is also moderately simple to compute, it is most likely the most efficient way to generate the desired random numbers.

A.2 REJECTION TECHNIQUE[2]

If the inverse function cannot be easily calculated, then the rejection technique should be considered. Suppose that the function, f(x), has a maximum value M where x varies over the range of interest from a to b. Random numbers are then chosen by the following two-step procedure.

- Select x from a uniform distribution on the interval (a, b)
- Select a second uniform random number, y, and accept the value x only if $y < [f(x)]/M$.

If x is rejected, then go back to the first step to select a new x and continue this procedure until some value of x is accepted. The probability of selecting x in the first step is $[1/(b-a)]\,dx$, while the probability of

acceptance at the second step is $f(x)/M$. Thus the x values will be generated with the desired probability $f(x)\,dx$.

The constant term $1/[M(b-a)]$ represents the efficiency of the rejection. Its reciprocal, $M(b-a)$, is the average number of trials the rejection technique will require to generate a single random number and is, therefore, linearly proportional to the computation time required. If $M(b-a)$ is very large, the rejection technique is too inefficient and a better technique should be sought.

The rejection technique need not be based on variables from a uniform distribution but can be developed from other distributions. For example the fact that

$$e^{-x^2/2} \leq e^{1/2} \cdot e^{-x}$$

can be used to develop a rejection technique for picking from a normal distribution. First select x from the exponential distribution e^{-x}. Then accept x if a second (uniform) random number

$$y < \frac{e^{-x^2/2}}{\sqrt{e} \cdot e^{-x}} = e^{-(x-1)^2/2} \quad .$$

The essential ingredient of the rejection technique is to find a second distribution function, $g(x)$, for which a selection procedure is known and such that $f(x) \leq C\, g(x)$. Selection of x from $g(x)$ is followed by acceptance if

$$y < \frac{f(x)}{C\, g(x)} \quad .$$

The average number of trials needed for an acceptance is C. Note that if $g(x)$ is close to $f(x)$, then C will be close 1 and the technique will be very efficient.

A.3 TRANSFORMATION

To simplify the derivation of inverse or rejection methods, it is best to transform the random variable into its simplest form. Thus, if one had $f(x) = g(\lambda x + \epsilon)$, one would first make the substitution, $y = \lambda x + \epsilon$, then

search for a technique for generating numbers from g(y). After generating a random number y, set $x = (y-\epsilon)/\lambda$ to get the desired random variable. In doing transformations correctly we must be careful to transform not just the function f(x) but the probability f(x) dx. Thus, properly, we have $f(x)\,dx = g(\lambda x + \epsilon)\,dx = g(y)\,dx = g(y)\,dy/\lambda$ as the substitution $y = \lambda x + \epsilon$ implies $dy = \lambda\,dx$. The correct normalized distribution for y is then $1/\lambda\ g(y)$. As a second example, assume $f(x)\,dx = 2x\,e^{-x^2}\,dx$. Try the transformation $y = x^2$. As $dy = 2x\,dx$, $f(x)\,dx = 2x\,e^{-x^2}\,dx = e^{-y}\,dy$. Therefore, selecting y from the exponential e^{-y} and taking $x = \sqrt{y}$ will give a random x from f(x).

A.4 COMBINATION OF RANDOM VARIABLES [2]

As a step beyond transformations, consider various combinations of random variables such as adding, subtracting, or multiplying two random numbers, taking the maximum or minimum of several random numbers, etc. The results of such combinations follow no intuitive pattern but must be worked out through the laws of probability. For example, the sum of two uniform random numbers has a triangular distribution, $f(x) = 1 - |x - 1|$ while the product has the distribution, $f(x) = -\ln x$. More complex examples seem even farther removed from simple rationality. If x and y are random numbers from the gamma distributions, $1/\Gamma(n)\ x^{n-1}\ e^{-x}$ and $1/\Gamma(m)\ y^{m-1}\ e^{-x}$, then $z = x/(x+y)$ has a beta distribution $\Gamma(m+n)/\Gamma(m)\Gamma(n)\ z^{n-1}\ (1-z)^{m-1}$. However, the beta distribution may also be obtained by taking $n + m - 1$ uniform random numbers, arranging them in increasing order, and selecting the nth number in the sequence. Thus, although combinations can be a very powerful method for transforming simple random variables into selections from other distributions, it is impossible to give guidelines or to arrive at a methodology for determining the proper combination needed to arrive at a desired distribution. The investigator must simply learn the frequently used combinations and must use his inventiveness when confronted with an unfamiliar distribution.

A.5 COMPOSITION TECHNIQUE[6]

Another method of general applicability is the composition technique. If the desired distribution can be written as a (generalized) integral over a family of density functions, then the sampling can be accomplished in a two-stage process. On the first step, a particular density function is selected from the family, and on the second step, the desired random number is drawn from the particular density function. In the usual application of this technique, the desired distribution is broken down into discrete parts, generally on separate intervals.

A.6 NUMERICAL METHODS

If no exact method can be derived, there is a numerical technique which can be used. This consists of generating the cumulative function, solving for its inverse numerically, tabulating the inverse, and then generating the random numbers from the tabulated data. If equal probability intervals are used in tabulating the inverse, then generation from the tabulated data can be quite fast. It does, however, require a certain amount of computer storage to hold the tabulation.

Improvements in the accuracy of numerical inverses can be made by using Chebyshev interpolating polynomials.[6] For some functions with long tails, the tabulated inverse must be replaced with some sort of approximating function in the tail of the distribution to achieve reasonable accuracy.

A.7 MARSAGLIA TECHNIQUE[3-5]

If a particular distribution is very central to a frequently used simulation program and the generation subroutine will be called a great many times to produce random numbers, it may be worthwhile to design a very fast selection procedure to reduce the computer time needed. A number of super-efficient techniques have been developed by G. Marsaglia.[3] These are based on composition methods where the function is expressed as the sum of three or more parts. The parts having highest probability are fast to select from and the parts difficult or slow to select from have very

small probability. In one of Marsaglia's methods, the function is broken into:

- A histogram
- A collection of saw-toothed functions where an efficient rejection technique selects from the 'almost-linear' distribution of each sawtooth.
- The tail of the distribution.

This method is very fast but requires moderate amounts of computer storage. In another method distributions are fitted to an approximation of the form $C(M + u_1 + u_2 + u_3)$, where M is a discrete variable and the u's are uniform variables. A small fraction of the time a more lengthy rejection procedure is needed to correct the error in the approximation. This method is fairly fast without great storage requirements.

These methods have been applied very successfully to the exponential and normal distributions. They do, however, require considerable effort in manhours to develop and thus should be applied to other distributions only when the payoff can justify it.

APPENDIX B

MIRAN

A MACHINE INDEPENDENT PACKAGE FOR GENERATING FROM DESIRED DISTRIBUTORS

APPENDIX B

MIRAN - A MACHINE INDEPENDENT PACKAGE FOR GENERATING UNIFORM RANDOM NUMBERS

B.1 GENERAL DISCUSSION

The standard technique for producing uniform random numbers on modern high-speed computers is an algorithm known as the multiplicative congruential method. This method is expressed mathematically as

$$R_{n+1} = \lambda \cdot R_n \text{ (modulo P)} .$$

Since the R's are integers ranging from 1 to P-1, successive real random numbers uniformly distributed from 0 to 1 are generated by dividing R_n by P. The properties of this technique as a random number generator (RNG) are highly dependent on the choice of the generator, λ, and the modulus, P. Unfortunately, there are many RNGs in current use which do not approximate randomness closely enough to be sufficient for all Monte Carlo calculations and, what is far worse, do manage to pass some of the simple tests for randomness. There are, however, several choices of λ and P which have been thoroughly tested, both theoretically[1] and through many years of actual use in Monte Carlo calculations, and which appear to be sufficiently random for general usage.

For reasons of convenience and efficiency, P is generally taken to be 2^m where m is the number of bits, excluding the sign bit, in a single word on the particular computer being used. The generation process starts with a fixed generator, λ, and a starting value, R_0. The full product from the multiplication of λ and R_0 would usually fill two computer words; however, the modulo P in the algorithm means that we only need the single word, R_1, comprising the low order half of the $\lambda \cdot R_0$ product. The random number generation is completed by converting R_1 to a real variable and

dividing by P. R_1 replaces R_0 in storage in the random number subroutine and the process is ready to begin anew.

In this sort of a process there have been two barriers to developing a Fortran RNG subroutine which would be independent of the particular computer for which it was designed. The first is the modulus P, which varies from computer to computer as the word length varies. [Choosing a universal value of P to fit the smallest computer is not a good solution as the properties of a RNG become less random as P is made smaller, to the extent that Coveyou and MacPherson[1] consider them questionable for $P = 2^{31}$ (IBM 360 series) and borderline for $P = 2^{35}$ (IBM 7090, Univac 1108, etc.).] The second problem is that the sign bit of R_1 may need to be cleared following the multiplication. Clearing the sign bit generally requires some trickery in Fortran which varies from computer to computer as the mode of representation (one's complement, two's complement, uncomplemented, etc.) of negative numbers varies.

The way around these obstacles is to use an explicit multiple precision representation. The integers and operations involved in the RNG algorithm are separated into component parts in such a way that all operations are kept within a single computer word and no overflows into the sign bit are made, thus avoiding the sign-clearing problem. Through multiple precision a sufficiently large modulus for good RNG properties may be used even though the actual computer word size is small. An initialization call must be made to convey to the RNG the maximum integer allowed on the particular computer being used so that it can set up an appropriate multiple precision representation.

The advantage of a RNG that is machine independent is simple: it greatly facilitates the exchange and checkout of Monte Carlo programs between different computers. The price paid for this advantage is also simple: it is a much slower method of producing random numbers. However, it is

still fast enough (several thousand random numbers generated in one second) that the time difference will not be noticed in most Monte Carlo applications.

B.2 CHOICE OF A SPECIFIC ALGORITHM FOR MIRAN

The work of Coveyou and MacPherson[1] has provided a thorough theoretical analysis of many commonly used RNGs. They show that the correlation properties of a RNG are strongly dependent on the modulus P. For values of $P = 2^{31}$ or 2^{35}, there must necessarily be a waviness or graininess to the joint distribution of two, three, and four consecutive random numbers that could lead to incorrect results for some Monte Carlo calculations. For $P = 2^{47}$, the departures from true randomness are small enough as to be negligible for practical calculations. Among the specific generators, λ, tested by Coveyou and MacPherson, there is one, $\lambda = 5^{15}$, which has good statistical properties and which may be easily produced by a machine independent subroutine. (In a subroutine designed for use on computers of varying word length, specifying a fixed 47-bit integer through data statements would be difficult. However, 5^{15} may easily be produced by multiplying 5's after the exact multiple precision representation needed has been established.) In addition the choice of $P = 2^{47}$ and $\lambda = 5^{15}$ has an added advantage: this particular choice of a RNG has seen long usage (several thousand hours on a CDC 1604 at Oak Ridge National Laboratory) in Monte Carlo computations without any apparent problems.

B.3 MULTIPLE PRECISION REPRESENTATION

In the basic algorithm used by MIRAN, λ and the R_n values will be 47-bit integers. This may exceed machine capacity. To keep all arithmetic operations from overflowing a single machine word, these integers are stored in an array wherein each word of the array constitutes a 'digit' in a representation of the integer to a particular base. This basis, called BASE, is chosen at execution time so that $(BASE)^2$ does not exceed the maximum integer allowed on the particular computer being used. Thus, for

example, on a machine with 35-bit words (unsigned), BASE would be 2^{17} and each 47-bit integer would be broken down into 3 words as follows:

47-bit Integer	Multiple Precision Representation	
$b_1 b_2 \ldots . b_{13} b_{14} \ldots . b_{30} b_{31} \ldots . b_{47}$	$+ 0 \ldots . 0\, b_1 \ldots . b_{13}$	word 3
	$+ 0 \ldots . 0\, b_{14} \ldots . b_{30}$	word 2
	$+ 0 \ldots . 0\, b_{31} \ldots . b_{47}$	word 1

Note that the 'digits' are stored in the array in 'reverse' order, i.e., word 1 is the least significant 17 bits of the number. Also, since 17 does not go evenly into 47, the last word contains only 13 bits.

Arithmetic in a multiple precision representation is carried out in the same manner as arithmetic is normally done by hand. The addition of two numbers, for example, is done digit by digit. When two 'digits', or words, are added there may be an overflow into the 18^{th} bit of the result. This must be detected, the overflow cleared out, and a carry of 1 added into the next higher 'digit'. Multiplication is slightly more complex. It is again carried out digit by digit and the resulting products are added, keeping them in appropriate columns, to get the final product. The multiplication of two 'digits' produces, of course, a two-digit product which is initially contained in a single computer word. This must be broken down into a high-order digit and a low-order digit with the high-order digit being added into the next higher column of the result. As each column is added, a carry over into the next higher column may be needed. Thus, in our example where three words were used for each integer, nine multiplies and several additions would be needed to form the six-word full product as schematized below.

$$
\begin{array}{cccccc}
 & & & d_3 & d_2 & d_1 \\
 & & & d_3' & d_2' & d_1' \\
 & & & & h_{11} & \ell_{11} \\
 & & & h_{21} & \ell_{21} & \\
 & & h_{31} & \ell_{31} & & \\
 & & & h_{12} & \ell_{12} & \\
 & & h_{22} & \ell_{22} & & \\
 & h_{32} & \ell_{32} & & & \\
 & & h_{13} & \ell_{13} & & \\
 & h_{23} & \ell_{23} & & & \\
h_{33} & \ell_{33} & & & & \\
\hline
S_6 & S_5 & S_4 & S_3 & S_2 & S_1
\end{array}
$$

where h_{ij} and ℓ_{ij} are the high and low order parts of the product of d_i and d_j'.

B.4 USE OF MIRAN PACKAGE

Initialization:

Before generating any random numbers, it is necessary to make an initialization call. This is done by the statement

CALL RANSET (MAXINT, NSTART)

where MAXINT is the maximum integer allowed on the computer (or compiler) being used. NSTART is the starting value, R_o, to be used in the random number sequence. If NSTART is less than or equal to 0, a default value of 2001 is supplied for NSTART. If NSTART is even, the next higher odd number will be used.

For example MAXINT = $2^{35}-1$ on a 1108, $2^{48}-1$ on a CDC-6600, etc. Good values for NSTART are any odd integer although frequent use of small odd integers is not recommended for calculations employing a relatively small number of random numbers.

The random numbers are generated in subroutine URAND which may be used as either a function subroutine or as an ordinary subroutine returning a value. Thus, either

```
CALL URAND(R)
        or
R = URAND(X)
```

will store a uniform random number in R. (Note that in the second form the same random number will also be stored in X. Thus, X must be a Fortran variable and not a constant.)

Limitations of MIRAN:

MIRAN will work on all computers where MAXINT is greater than 1023 and less than 2^{94}. (These limits are practical and not theoretical and could be extended if it were ever necessary.)

B.5 MIRAN PROGRAM DETAILS

The Fortran listings of the two MIRAN routines URAND and RANSET are presented in Figures B-1 and B-2. The accompanying logic flow is detailed in Figures B-3 and B-4. Additional explanation of the last step in the URAND logic is provided below.

The two subroutines URAND and RANSET communicate through a labelled common, MIRNG which contains

RAN(10) - An array containing the 'digits' of the current (or last) multiple precision random integer

```
      REAL FUNCTION URAND(FRAN)
      COMMON /MIRNG/ RAN(10),GEN(10),NWRD,BASE,MOD,FBASE,FMOD
      DIMENSION SUM(10)
      INTEGER RAN,GEN,BASE,CARRY,SUM,PROD,HPROD
      DO 30 IS=1,NWRD
   30 SUM(IS)=0
      DO 1 IG=1,NWRD
      N2=NWRD-IG+1
      DO 1 IR=1,N2
      IS=IR+IG-1
      PROD=RAN(IR)*GEN(IG)
      HPROD=PROD/BASE
      LPROD=PROD-HPROD*BASE
      SUM(IS)=SUM(IS)+LPROD
      IF (IS.LT.NWRD) SUM(IS+1)=SUM(IS+1)+HPROD
    1 CONTINUE
      N2=NWRD-1
      DO 5 IS=1,N2
      CARRY=SUM(IS)/BASE
      SUM(IS)=SUM(IS)-CARRY*BASE
      SUM(IS+1)=SUM(IS+1)+CARRY
    5 CONTINUE
      SUM(NWRD)=SUM(NWRD)-MOD*(SUM(NWRD)/MOD)
      DO 20 IS=1,NWRD
   20 RAN(IS)=SUM(IS)
      FRAN=SUM(1)
      DO 10 IS=2,NWRD
   10 FRAN=FRAN/FBASE+SUM(IS)
      FRAN=FRAN/FMOD
      URAND=FRAN
      RETURN
      END
```

Figure B-1. Fortran listing of URAND

```
      SUBROUTINE RANSET(MAXINT,NSTRT)
      COMMON /MIRNG/ RAN(10),GEN(10),NWRD,BASE,MOD,FBASE,FMOD
      INTEGER RAN,GEN,BASE,CARRY,REM
      MAXI=MAXINT/4
      IB=0
      BASE=1
   99 IF (BASE.GT.MAXI) GO TO 100
      BASE=BASE*4
      IB=IB+1
      GO TO 99
  100 BASE=2**IB
      FBASE=BASE
      NWRD=47//IB+1
      REM=47-IB*(NWRD-1)
      MOD=2**REM
      FMOD=MOD
      DO 101 N=1,10
      RAN(N)=0
  101 GEN(N)=0
      GEN(1)=5
      DO 200 I=1,14
      CARRY=0
      DO 190 N=1,NWRD
      GEN(N)=GEN(N)*5+CARRY
      CARRY=0
      IF (GEN(N).LT.BASE) GO TO 190
      CARRY=GEN(N)/BASE
      GEN(N)=GEN(N)-BASE*CARRY
  190 CONTINUE
  200 CONTINUE
      NSTART=NSTRT
      IF (NSTART.LE.0) NSTART=2001
      NSTART=2*(NSTART/2)+1
      DO 300 N=1,NWRD
      NTEMP=NSTART/BASE
      RAN(N)=NSTART-NTEMP*BASE
  300 NSTART=NTEMP
      RETURN
      END
```

Figure B-2. Fortran listing of RANSET

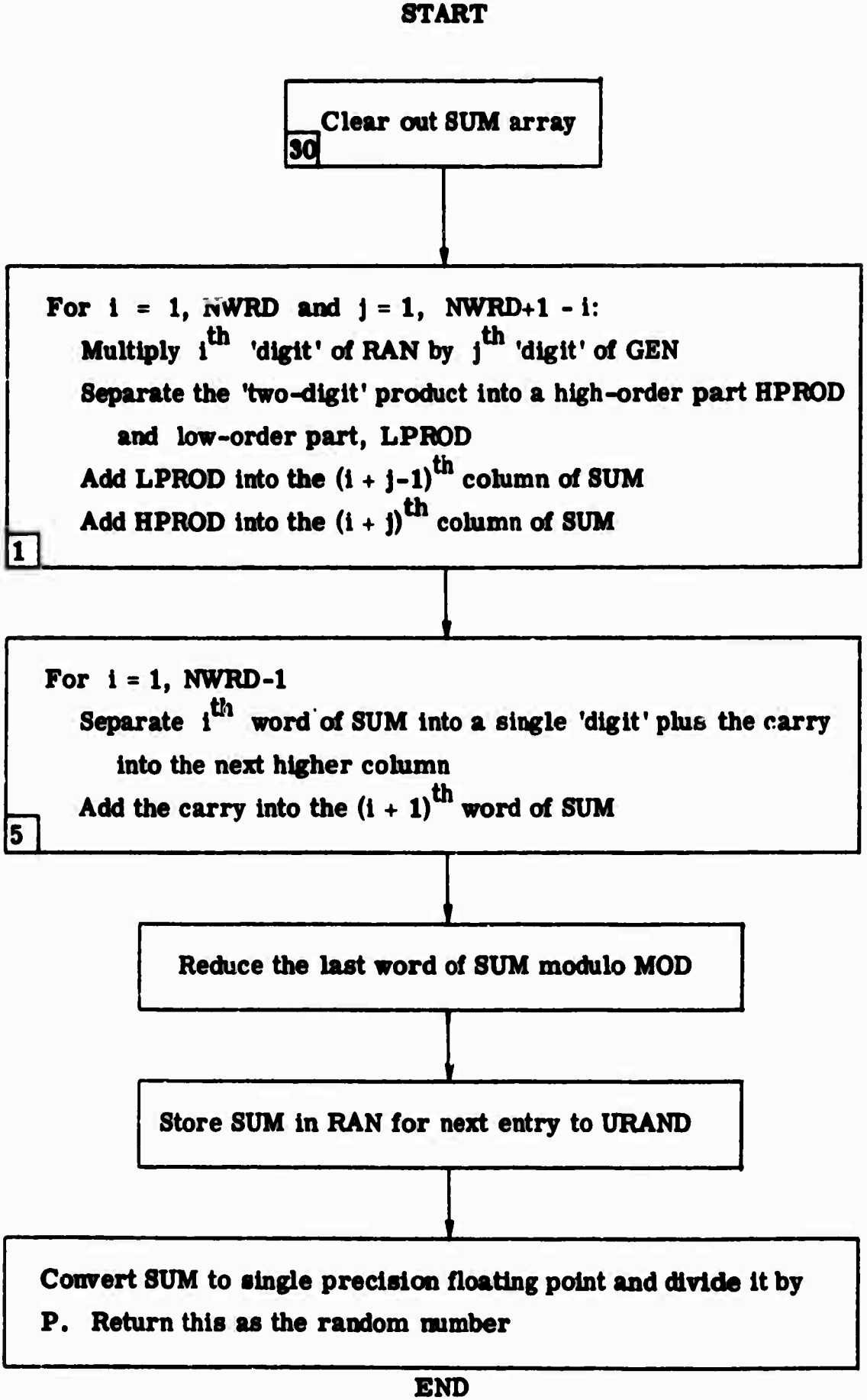


Figure B-3. Logic flowchart for URAND

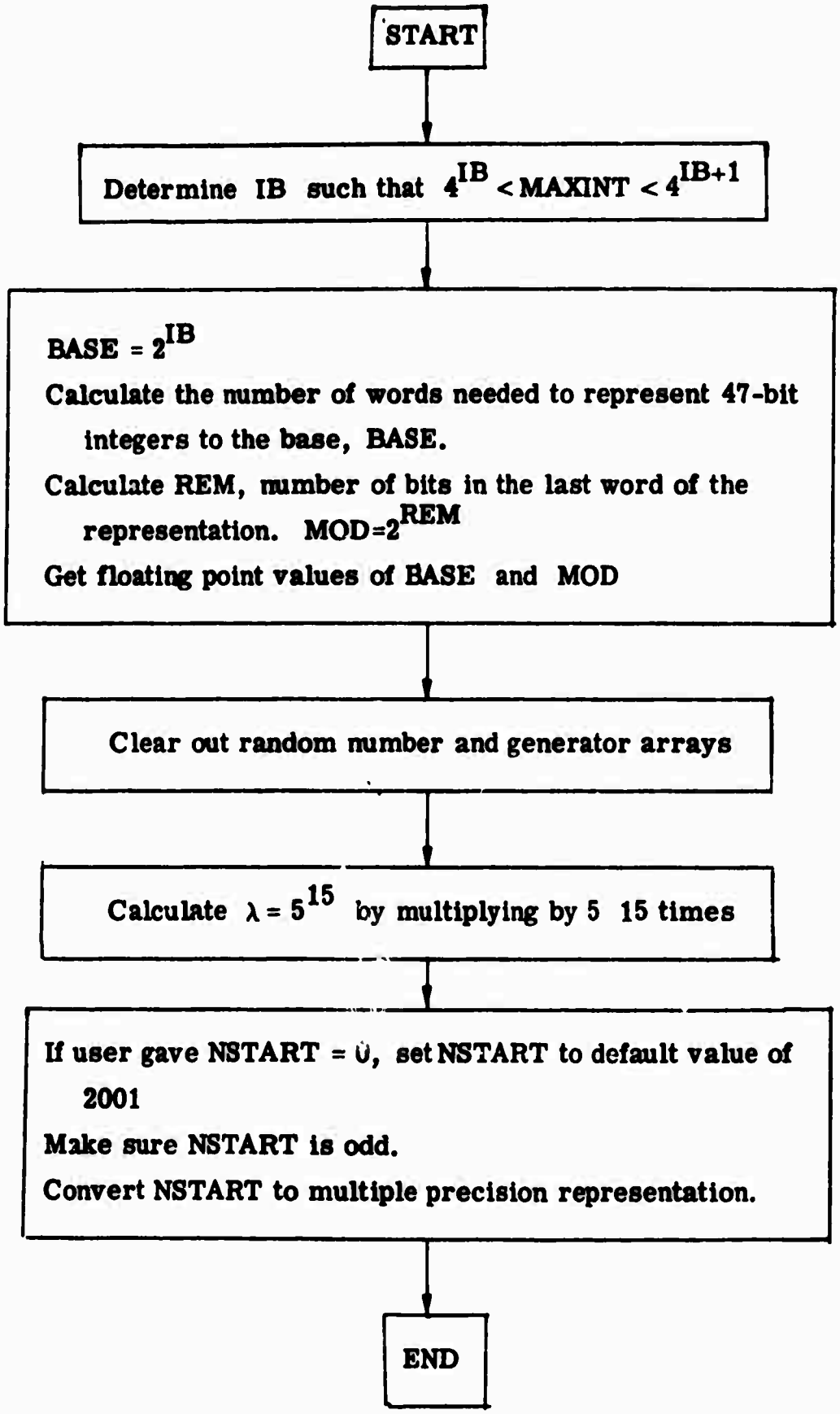


Figure B-4. Logic flow chart for RANSET

GEN(10)	- An array containing the generator $\lambda(= 5^{15})$ in multiple precision representation
NWRD	- The number of words used in the multiple precision representation of an integer
BASE	- The base used in the multiple precision representation
MOD	- The maximum value of the highest order 'digit' in the multiple precision representation
FBASE	- Floating point value of BASE
FMOD	- Floating point value of MOD

RAN, GEN, NWRD, and NBASE are Fortran integers; FBASE and FMOD are Fortran real quantities.

An alternative method (unfortunately, not machine independent) of giving the routine a starting value is to save the array RAN at the end of a run and to restore RAN at the start of the new run (just after the RANSET call).

In the last step of the URAND flow the objective is conversion of the multiple precision integer random number R to a floating point random number X between 0 and 1. The multiple precision integer produced by the random number algorithm is represented by the 'digits' $r_1, r_2, \ldots\ldots, r_n$ (remember that r_1 is the lowest order digit. Thus,

$$R = r_1 + (BASE)\cdot r_2 + (BASE)^2 \cdot r_3 + \ldots\ldots + (BASE)^{N-1} \cdot r_N \ .$$

Notice that we have, from the manner in which N and MOD were established,

$$P = (BASE)^{N-1} \cdot MOD \ .$$

The uniform random number desired is given by R/P. Thus we have,

$$X = \frac{R}{P} = \frac{r_1}{(BASE)^{N-1} \cdot MOD} + \frac{r_2}{(BASE)^{N-2} \cdot MOD} + \frac{r_3}{(BASE)^{N-3} \cdot MOD}$$

$$+ \ldots + \frac{r_{N-1}}{BASE \cdot MOD} + \frac{r_N}{MOD}$$

$$= \frac{1}{MOD}\left(r_N + \frac{1}{BASE}\left(r_{N-1} + \cdots \frac{1}{BASE}\left(r_2 + \frac{1}{BASE} \cdot r_1\right)\ldots\right)\right)$$

Starting from the right it is easy to compute this iteratively.

B.6 FIRST 100 RANDOM NUMBERS PRODUCED BY MIRAN

For checkout purposes, Table B-1 lists the first 100 random numbers produced by MIRAN when the default value of NSTART, 2001, is used as the starting random number.

TABLE B-1

First 100 Random Numbers Produced by Machine-Independent Random Number Generator

.4338977	.7498470	.9903279	.8066061	.9646561	.4108350	.5079340	.2304957	.8253787	.6423580
.3726977	.8990164	.5649909	.6669998	.8944935	.0315737	.6045930	.2886992	.7004357	.0770901
.7623080	.8476278	.5594711	.2497678	.2724822	.1385819	.1319806	.8203205	.2988024	.9646986
.1032452	.2195750	.0736062	.0596241	.4543268	.6996527	.5664641	.4486428	.8878830	.8846283
.6556108	.3540241	.4236103	.1514489	.6888500	.9272675	.4724877	.6947799	.4599674	.4813375
.7548023	.4671775	.9455987	.1946915	.2781693	.7066840	.2893777	.6862300	.9028574	.3585868
.6711822	.1015501	.0777647	.5482782	.7349975	.0763710	.0634225	.0418916	.6368102	.6480426
.8594980	.7903770	.6175994	.8909496	.3714798	.1235697	.0322795	.4017063	.7728325	.0014033
.3249444	.5456800	.6629544	.4225348	.5104638	.6921146	.2768275	.3440949	.9040403	.5348431
.2713845	.6928810	.3603209	.2550091	.2913790	.0058372	.9934161	.4194589	.9584109	.8090146

APPENDIX C

REFERENCES AND ABSTRACTED BIBLIOGRAPHY

1. Coveyou, R. R., and R. D. MacPherson, "Fourier Analysis of Uniform Random Number Generators," Journal of the ACM, 14 pp. 100-119, 1967.

A method of analysis of uniform random number generators is developed, applicable to almost all practical methods of generation. The method is that of Fourier analysis of the output sequences of such generators. With this tool it is possible to understand that predict relevant statistical properties of such generators and compare and evaluate such methods. The results of many such analyses and comparisons are given. The performance of these methods as implemented on differing computers is also studied. The main practical conclusions of the study are: (a) Such a priori analysis and prediction of statistical behavior of uniform random number generators is feasible. (b) The commonly used multiplicative congruence method of generation is satisfactory with careful choice of the multiplier for computers with an adequate ($\geq\sim 35$ bit) word length. (c) Further work may be necessary on generators to be used on machines of shorter word length.

2. Kahn, H., Applications of Monte Carlo, Rand Corp., AEC-3259, USAEC, April 1964.

A classic publication in the field of Monte Carlo methods that describes general Monte Carlo methods, random number generation schemes and variance reduction techniques. The volume is divided in two parts. Part I describes basic techniques with random numbers (such as fundamental random number generation techniques) and Part II details several variance reduction schemes. The general areas of application addressed are problems in radiation transport.

3. MacLaren, M.D., G. Marsaglia, and T. A. Bray, "A Fast Procedure for Generating Exponential Random Variables," Communications of the ACM, 7, May 1964.

A very fast method for generating exponential random variables in a digital computer is outlines. A detailed flow diagram and required tables are provided.

4. Marsaglia, G, and T. A. Bray, "A Convenient Method for Generating Normal Variables," SIAM Review, 6, 1964.

A very fast yet small Fortran routine for generating normal random variables in terms of a sequence of random variables uniform over [0, 1] is presented. A random variable X is generated in terms of uniform variables $U_1, U_2, \ldots$ in the following way: 86 percent of the time, $X=2(U_1+U_2+U_3 - 1.5)$, 11 percent of the time, $X = 1.5 (U_1+U_2 - 1)$, and the remaining 3 percent uses a complicated procedure.

5. Marsaglia, G., M.D. MacLaren, and T. A. Bray, "A Fast Procedure For Generating Normal Random Variables," Communications of the ACM, 7, 1964.

A technique for generating normally distributed random numbers is described. It is faster than those currently in general use and is readily applicable to both binary and decimal computers.

6. National Bureau of Standards Applied Mathematics Series 55, June 1964. Handbook of Mathematical Functions, Numerical Methods, pp. 949-953.

This section of the handbook reviews various methods of generating random numbers including the rejection and composition methods. Also presented are specific techniques for various discrete and continuous distributions such as the normal and exponential distributions.

7. Spanier, J., and E. M. Gelbard, Monte Carlo Principles and Neutron Transport Problems, Addision Wesley Publishers, 1969.

This is one of the more recent comprehensive references on Monte Carlo methods as applied to radiation transport problems. Basic fundamentals of Monte Carlo are first reviewed. Next the concepts of discrete and continuous random walks are introduced followed by a discussion of variance reduction techniques. Finally, advanced concepts and applications to radiation transport are presented.